初めての人の

C言語入門

JTM企画株式会社／
●Writer 荒瀬 遙

パソコン初心者でもわかる
C言語入門書の決定版!!

本書を読むためには、
なんの予備知識もいりません。
BASICなど、ほかの言語を
知っている必要もありません。
簡単なワープロ文書なら作れる、
といった程度で十分です。
本書の前半部では、
C言語の背景となる部分や関連事項を
解説しています。
ここで、C言語のしくみや考えかた、
プログラミングの概略がわかります。
後半部でようやく具体的な
プログラムを紹介し、
C言語そのものの特徴を解説しています。
このように本書は、
まったくのC言語入門者、初心者が、
少し本格的に
取り組んでみたいというところまでの
橋渡し的な役割をになうものです。
ざっと読み進めるだけでもC言語とは
どういうものかはわかりますが、
できれば実際に操作し、
プログラミングのテクニックに
挑戦してください。

西東社

全国学校図書館協議会選定図書

初めての人の

C言語入門

JTM企画株式会社／
●Writer 荒瀬（あらせ） 遙（はるか）

西東社（せいとうしゃ）

はしがきにかえて

本書の企画主旨とC言語

あなたは、プログラミング言語に興味をお持ちでしょうか？　あるいは、
何かひとつ、プログラミング言語を覚えたい——
BASIC（ベーシック）はだいたいわかったから、別の新しい言語に挑戦してみたい——
コンピュータは敬遠したいけど、仕事で必要にせまられている——
ということでしょうか？

C言語は、近年、ますます脚光を浴びているプログラミング言語です。さまざまな雑誌でとりあげられ、関連書籍も数多く発売されています。

しかし、本書を手にされたあなたの動機がなんであるにせよ、C言語はどうもむずかしそうだというのが共通の思いではないでしょうか。

これまでに刊行されたC言語入門書の多くは、Ⓐビギナー向けでわかりやすいが、C言語そのものの姿やしくみが見えてこない、Ⓑわかる人向けのテクニックなどの解説が多く、初心者にはむずかしい、のどちらかであるように思われます。そこで本書は、そのⒶとⒷのすきまを埋めるための“C”言語入門として企画されたものです。

本書を読むためには、なんの予備知識もいりません。BACICなど、ほかの言語を知っている必要もありません。簡単なワープロ文書なら作れる、というくらいで十分なのです。

本書に登場する田川氏も、そんなパソコン初心者のひとりです。「コンピュータは人にまかせておこう」という信条だった田川氏が、仕事の必要にせまられてC言語を学ぶことになりました。そんな田川氏とともに、あなたもC言語の世界に触れてみてください。

本書の前半部では、プログラムをほとんど紹介していません。PART1およびPART2では、C言語の背景となる部分や関連事項の解説に重点を置きました。初めてプログラムが出てくるのは、PART3に入ってからです。そして、後半部でC

言語そのものの特徴を解説しています。

これも、いきなりプログラムから入るよりも、C言語のしくみ、考えかたやプログラミングの概略について、あらかじめ理解しておいたほうがよいという配慮からきています。

実際の複雑なプログラミングのテクニックについては、本書の範囲を超えています。既刊の書籍などを参考にたくさんプログラムを組んでみて、実地で体得していってください。ただ、C言語特有の構造体やポインタ、また、そもそも関数とか変数とはどういうものかについて、わかりやすく説明したつもりですから、今後の学習のための指針にはなるはずです。

本書を使いこなすために

C言語を使うためには、パソコンなどのハードとC言語ソフトが必要です。本書では前述した企画主旨から、特定のハードやソフトに限定していません。

パソコンは、最も普及しているOS（オペレーティング・システム）である、MS-DOSが使える機種ならすべて使用することができます（UNIXが使用できるコンピュータでもよい）。

C言語ソフトは、さまざまな価格とタイプのものが販売されています。代表的なものについては特徴と簡単な操作方法を解説しておきましたから、選ぶときの参考にしてください。

本書は、ひととおり読むだけでもC言語についての知識が得られるものですが、実際に操作しながらのほうが覚えやすいのは当然です。できればC言語ソフトをどれか購入し、入力→実行しながら読み進めてください。例にあげたプログラムは簡単なものですが、どれも実際に使えるものばかりです。また、PART6には仕事別の完成品プログラムを紹介しておきましたから、実際に使用、あるいは改良してみてください。

JTM 企画株式会社／荒瀬　遙

もくじ

PART 1 C言語を知ろう 9〜34

PART 2 C言語プログラミングの基礎知識 35〜58

PART●3 いよいよプログラムだ 59〜82

PART・4 プログラミングのテクニック—1 83〜120

PART・5 プログラミングのテクニック—2 121〜158

PART・6 すぐに役立つ仕事別プログラム 159〜185

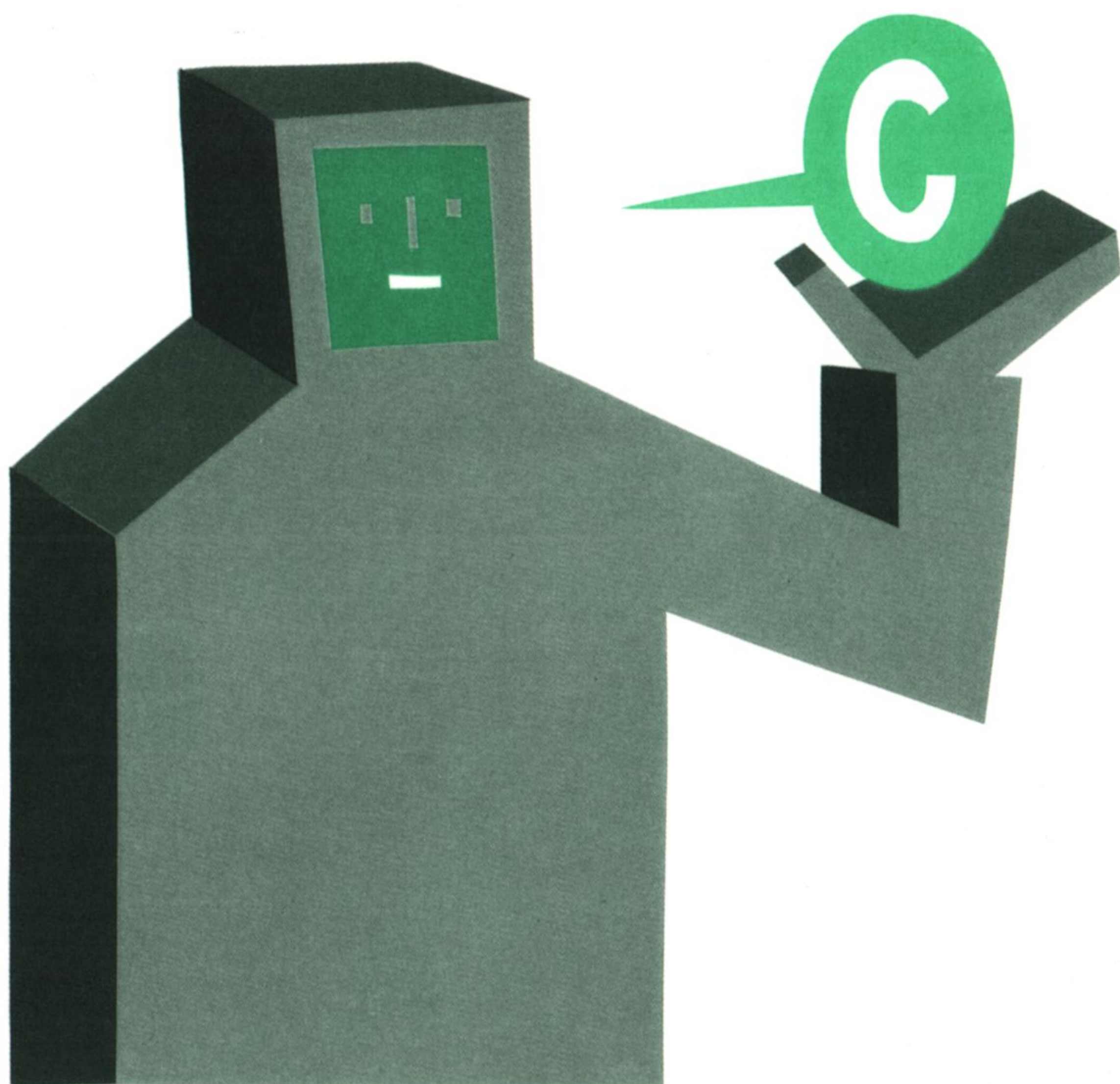
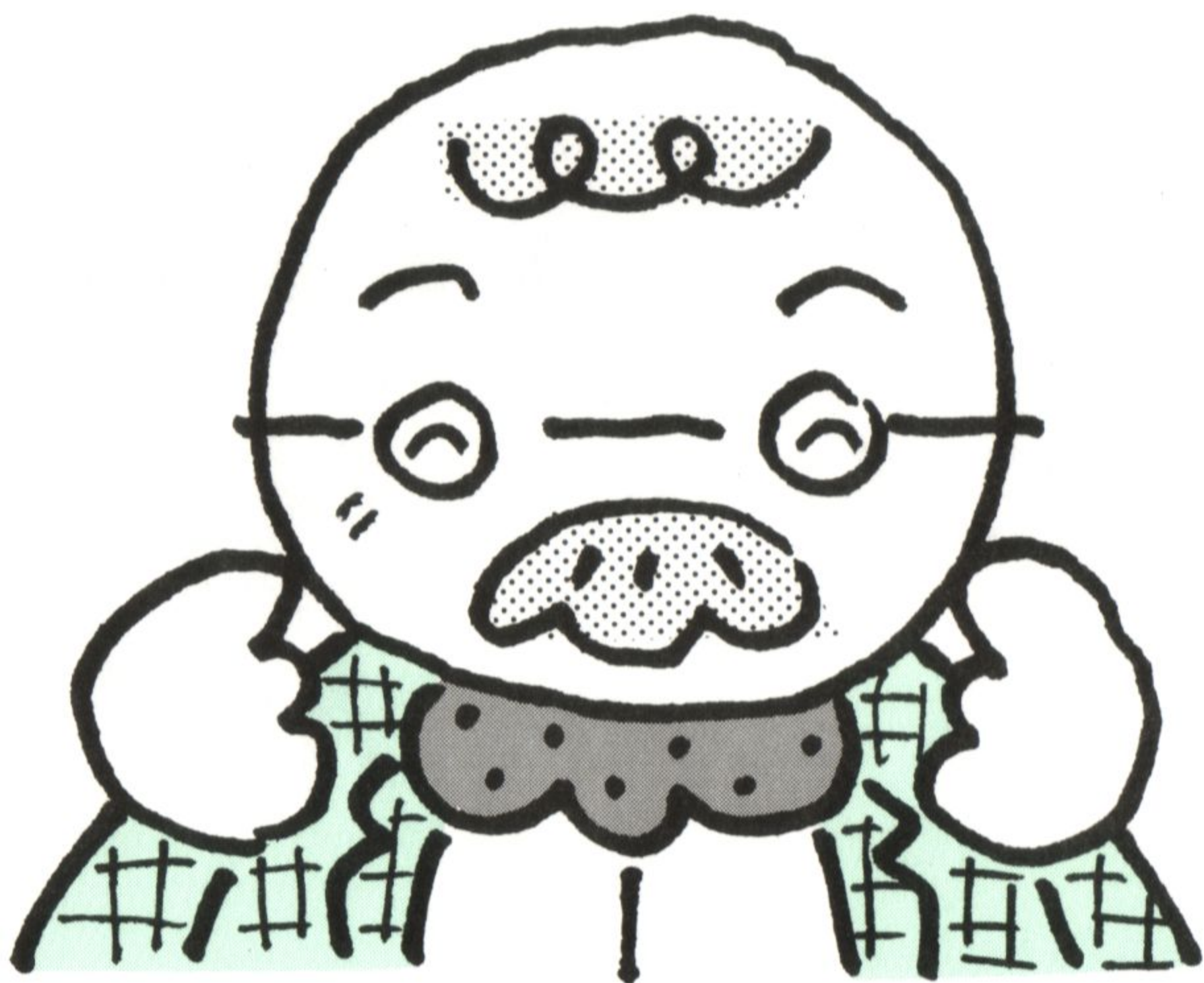
■ふろく

◉一覧表

◉コラム

写真●
平野時義
イラスト●
伊藤博幸
入江友芳
すがわら　けいこ
ネモト　ヤスオ

PART 1

C言語を知ろう……

田川氏、姪のもとを訪ねる。

自動車部品関係の仕事をしている某大企業の部長田川氏は、ある日曜日、姪でプログラマの石室玲子を訪ねた。

玲子　あら、おじさん、お久しぶり。

田川　玲ちゃん、ちょっとお願いがあるんだよ。実はＣ言語とやらを、至急教えてくれないか。

玲子　どうしたの、藪から棒に？　この前までは、コンピュータには興味がないっていってたんじゃないの。「あんなわけのわからん機械」って。

田川　いや、それはいまでもそう思っとるんだが……うちの会社にもパソコンが入って、若い連中が一太郎だのロータスだのを使っていたんだよ。ところが、いちばんパソコンのわかる人間が、先月、突然会社をやめてしまったんでね、あわてているんだよ。

玲子　あら、それはたいへんね。

田川　そいつが、仕事に使うプログラムをいろいろ作ってくれていたらしいんだが、それがＣ言語で書いてあるというんだ。ほかの者は「BASIC（ベーシック）なら少しはわかるけれどＣ言語はちょっとわからない」なんて頼りないかぎりで、私も勉強しなくちゃいけないなあと思ってね。これから、いろいろと教えてくれないか？

玲子　いいわよ。ところで、おじさん、コンピュータのことはどのくらい知っているの？　たしか、パソコンなんて触ったこともなかったんでしょ。

田川　そうなんだよ、ところが、今度の事件だろ。とりあえず部の女の子に頼んで、「一太郎」と「Lotus1-2-3」を教えてもらっているところだよ。一太郎では簡単な文章なら書けるようになった。でもLotus1-2-3はむずかしいね。表を作るくらいはできるけど、マクロなんていわれるとお手上げだね。

玲子　ちょっとの間にだいぶ勉強したみたいね。一太郎を使えるようになるまでけっこう時間がかかる人もいるから、1か月もたたないうちにそれだけできるようになったら上出来よ。

田川　しかし、一太郎とかLotus1-2-3まではまだいいんだが、プログラムとなると皆目見当がつかなくて、玲ちゃんのことを思い出したというわけだ。

玲子　プログラムは、ちょっとたいへんかもしれないけれど、だいじょうぶ？

田川　うーん、とにかくC言語がどんなものかくらいは知っておきたいし、C言語で簡単なプログラムを書けるようになりたいんだ。お礼に、フランス料理のフルコースでもおごらせてもらうよ。

玲子　それじゃあちょっと安いわね。お正月の2泊3日スキーご招待、ぐらいはやってもらいたいわね。

田川　わかった、わかった……。C言語の勉強をするのもたいへんだなあ。

玲子　え？　なあに。

田川　いや、なんでもない、なんでもない。ところでC言語というけれど、このCは何かの略なのかい？

玲子　C言語はComputerのCだとか、Cなんとかかんとかの略だと思っている人が多いけれど、C Languageというのが正しい名前よ。だから、日本語に訳してC言語。はじめにBCPLという言語が作られ、次にB言語というのができ、B言語の次だからC言語になったのよ。

田川　冗談だろう？

玲子　いいえ、これはホントの話。でも、C言語はまさにコンピュータ言語というのにぴったりの言語かもね。シンプルだし、覚える部分は少なくてすむし。

田川　そうか、よし、これから、C言語を覚えるためにがんばるぞ。

玲子　おじさん、いきなりC言語というのもたいへんだから、その前にコンピュータのこととか「一太郎」「Lotus1-2-3」などの市販ソフトとか、プログラムについて、C言語とからめながら説明していくわね。

パソコンの世界とC言語

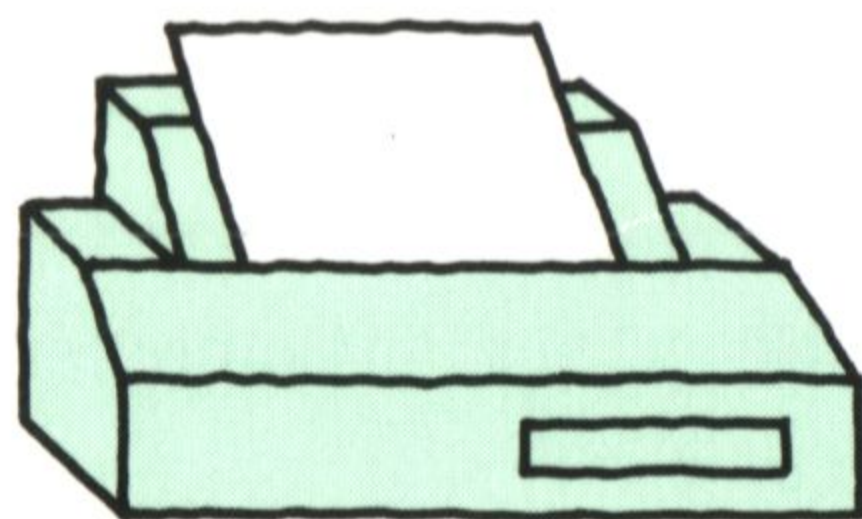
パソコンとハードウェア

6 パソコン

C言語を使うためには、パソコンの本体、ディスプレイ（画面）、キーボード、フロッピィディスク装置があれば十分です。これに、RAMディスクやハードディスクがあればより便利でしょう。

パソコンにもいろいろな機種があり、日本電気のPC-9800シリーズ、東芝J-3100シリーズ、富士通FMRシリーズ、エプソンPC-286シリーズなどさまざまな製品があります。C言語は、これらのすべての機種上で使うことができます。ここにあげた名前は、すべて16ビット・パソコンと呼ばれるものですが、このほかに、32ビット・パソコンや、ワークステーションと呼ばれるコンピュータ、大型コンピュータでもC言語を使うことが可能です。

●本体
パソコンの頭脳部と記憶装置。ソフトの命令を解読し、指令を与える

●ディスプレイ
入力した文章を表示する画面。各種操作もこの画面上で行う

●プリンタ
できあがった文書を印刷する

●キーボード
文字を入力して漢字カナまじり文に変換させたり、いろいろな特殊キーを使ってソフトに命令する

●フロッピィディスクドライブ（2台）

▲C言語を使用するためのパソコンの機器類

フロッピィディスク

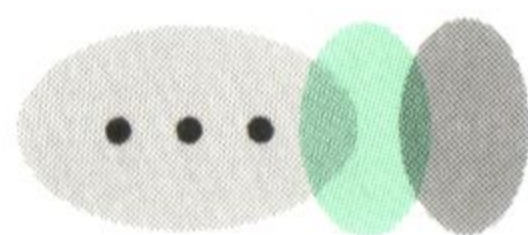
フロッピィディスクには8インチ、5インチ、3.5インチの3種類の大きさがあり、使用するパソコンのディスクドライブ（装置）によって使い分けます。主に使われているのは5インチ2HDと呼ばれるもので、記憶容量は1メガバイト（1MBと書き表す）。漢字なら約52万個の文字を記録することができる大きさです。8インチ2Dのフロッピィディスクも、同じく1MBの容量を持っています。5インチ2DDのフロッピィディスクもありますが、これはPC-9800Fなどの機種で使用されていたもので、現在はあまり使われていません。こちらは容量が640キロバイト（KB）で、漢字約32万字を記録できます。

3.5インチのディスクは、最近多く使われるようになってきたもので、ラップトップ・パソコンやデスクトップ・パソコンの一部に使われています。プラスチック製の硬いケースに入っているので、扱いやすいことが利点です。3.5インチ2HDは1MB、3.5インチ2DDは640KBです。

ハードディスクとRAM（ラム）ディスク

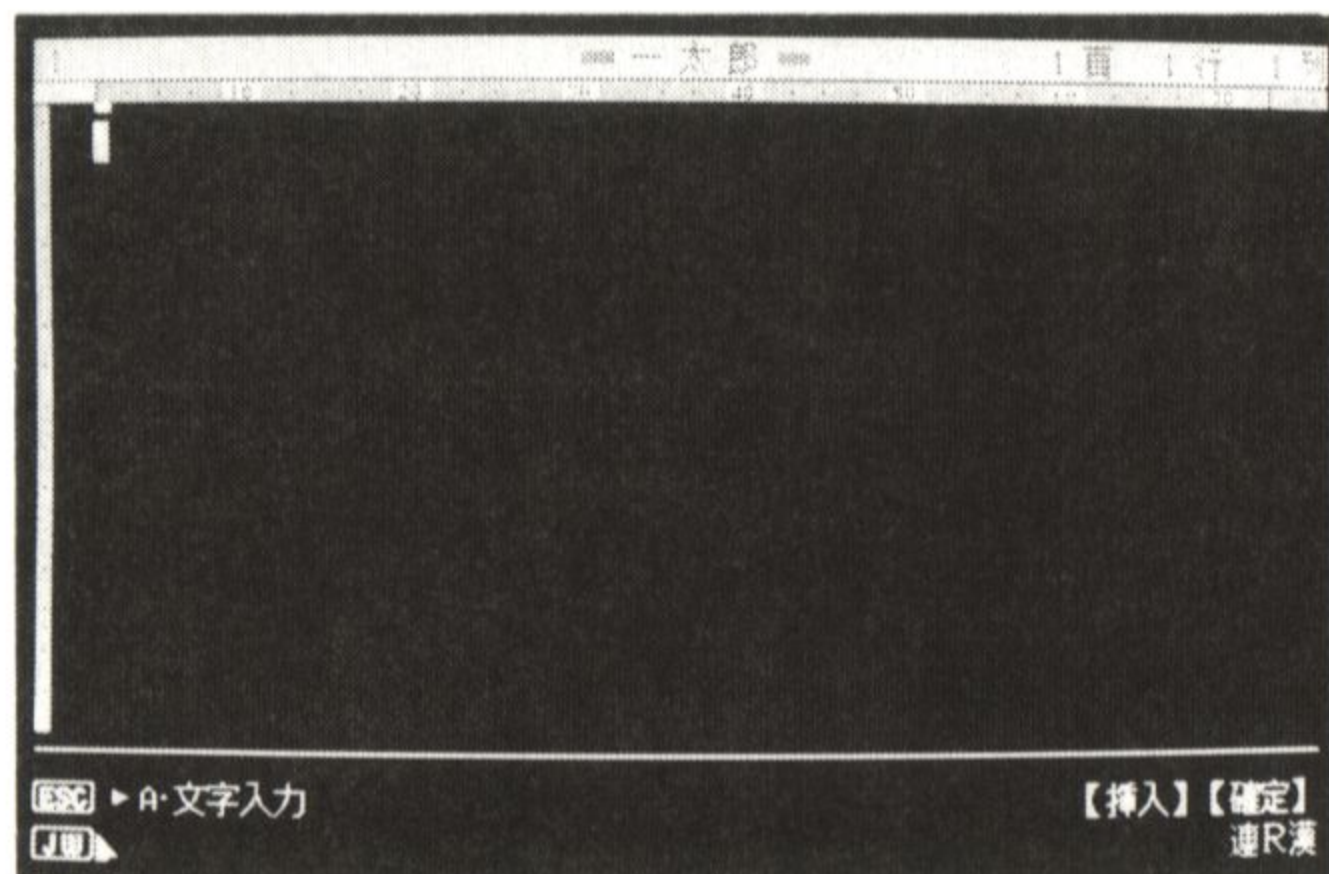
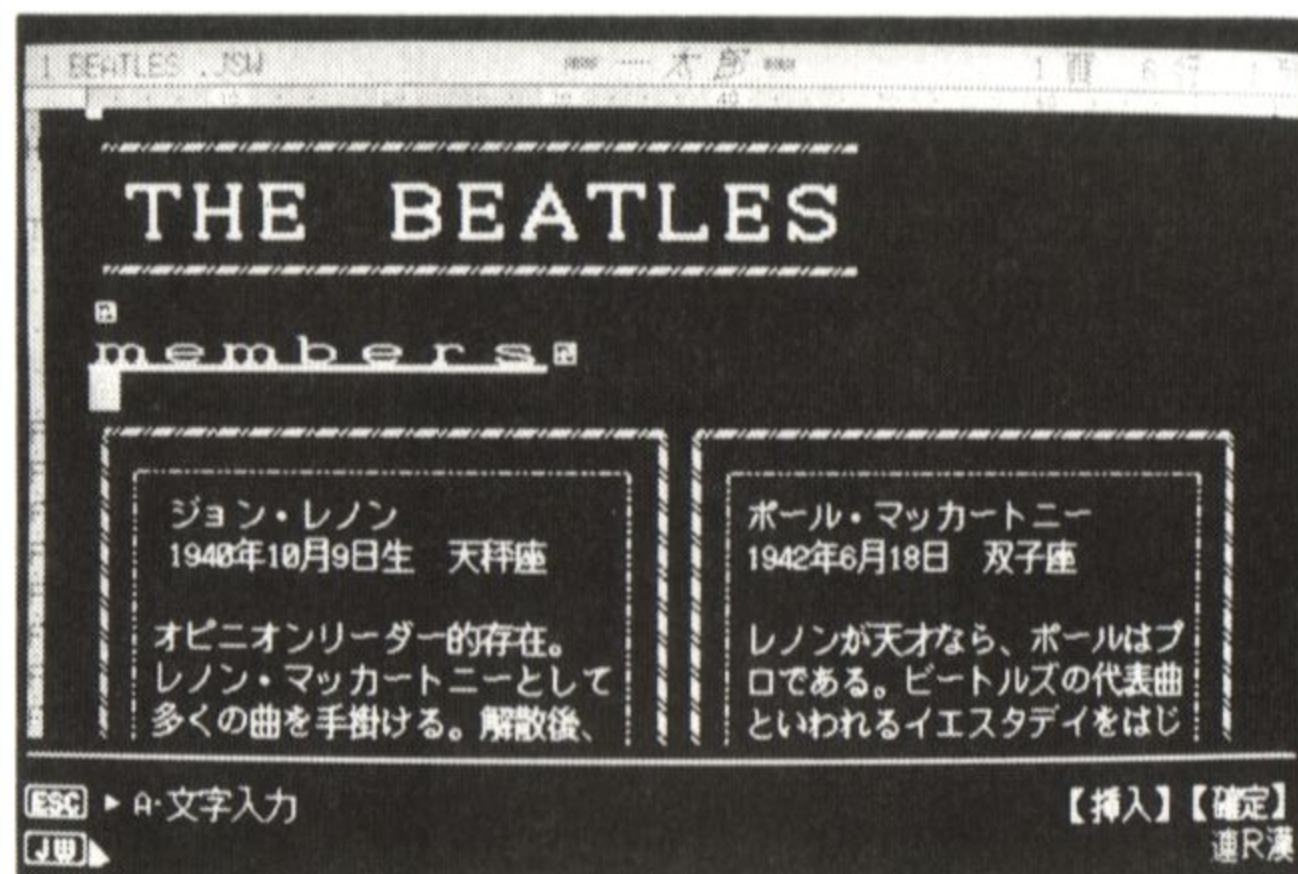
フロッピィディスクでは、ソフトを起動したりデータの出し入れをするのに時間がかかるうえ、何枚ものフロッピィをそのつど交換する必要があり、操作もめんどうです。たくさんのソフトやデータを利用する場合は、ハードディスク（固定ディスク装置）やRAMディスクを使用すると便利でしょう。

ハードディスクとは、四角い箱の中に、フロッピィディスクの何十倍もの記憶領域を持つディスクが複数枚セットされた装置です。パソコンに接続し、いろいろなソフトを登録しておけば、キー操作だけでそれらを使い分けることが可能です。また、ディスクドライブとしてもフロッピィディスクに比べてはるかに高速で、いろいろなソフトを動かしたり、データを処理することができます。最近ではハードディスク内蔵タイプのパソコンが多く発売され、外付タイプのものに比べ、取り扱いも便利になりました。

RAMディスクも、仮想的なディスクドライブとして、ハードディスクと同様に使用することができます。ただし、ハードディスクは電源を切っても登録されたデータは消えませんが、RAMディスクの場合は、電源を切ると蓄えたデータが消えてしまいます。そこで、RAMディスクをディスクドライブとして使用したあとは、最後に、必要なデータをフロッピィディスクなどに保存しなければなりません。RAMディスクは日本語変換のための辞書や、作業領域を一時的に設ける場所として使われています。

市販のパッケージソフト

ワープロソフト「一太郎」（ジャストシステム）

パッケージソフトとは、そのままですぐに使えるよう商品化されたソフトウェアのことです。ソフトのフロッピィとマニュアルなどがセットされ、パッケージに入れて売られていることから、このような呼び名がつけられました。ワープロといえば「一太郎」といわれるように、一太郎はパソコン用のパッケージソフトのなかで、最も良く知られたワープロソフトです。

簡単にいうと、一太郎のフロッピィディスクをディスクドライブに入れてパソコンを起動すると、写真のような画面が現れ、パソコンをワープロとして使用できるようになります。つまり、この状態で文章を書いたり、書いた文章を印刷したりできるのです。

▲一太郎の文書作成画面

▲作成した文書例

表計算ソフト「Lotus1-2-3」（ロータス）

パッケージソフトのもうひとつの代表は、表計算ソフト「Lotus1-2-3」です。表計算ソフトというのは、文字どおり表形式のデータを扱うもので、各種の計算やグラフ作成などの機能を持っています。たとえば、例に示した交通費の精算書をLotus1-2-3で作ると、次ページ画面写真のようになります。このように、表の中で計算式を定義しておくと、数字を一部変更した場合でも瞬時のうちに再計算され、合計や平均値などが自動的に修正されるという、たいへん便利なものです。

出張旅費精算書 年 月 日提出

社長	総務	課長	主任

月/日	項目（支出した順に書いて下さい）	金額 交通費	交際費	通信費・その他
/		円	円	円
/				
/				
/				
/				
/				
/				
/				
/				
/				
小計		①	②	③

日当 日数	単価	宿泊費 泊数	単価	金額	合計
	円		円	円	⑤ 円
合計④ 円					

Ⓐ 仮払金	Ⓑ 旅費合計（①+②+③+④+⑤）	差引過不足額（Ⓐ−Ⓑ）
円	円	円

出張区域	期間	氏名
	月 日 月 日 日間	㊞

日本法令 販売6−1

▲交通費精算書（日本法令）

A13:

	A	B	C	D	E
1			出張旅費精算書		
2	月日	項目	交通費	交際費	通信費その他
3	5/10	飛行機	47000		
4	5/10	地下鉄	180		
5	5/10	ホテル代			12000
6	5/11	地下鉄	220		
7	5/11	接待		24000	
8	5/11	タクシー	2430		
9	5/11	ホテル代			12000
10	5/12	タクシー	1500		
11	5/12	電話			1480
12	5/12	地下鉄	180		
17	小計		51510	24000	25480
18	合計				100990

▲交通費精算書の画面

	B	C	D	E
1		出張旅費精算書		
2	項目	交通費	交際費	通信費その他
3	飛行機	47000		
4	地下鉄	180		
5	ホテル代			12000
6	地下鉄	220		
7	接待		24000	
8	タクシー	2430		
9	ホテル代			12000
10	タクシー	1500		
11	電話			1480
12	地下鉄	180		
17		@SUM(C3..C16)	@SUM(D3..D16)	@SUM(E3..E16)
18				@SUM(C17..E17)

▲設定した計算式

6 その他のパッケージソフトとプログラム

パソコンは職場、学校のほとんどに導入され、家庭にも徐々に普及しつつあります。パソコン通信も、仕事にホビーにと盛んになってきています。パッケージソフトには、ワープロ、表計算のほか、データベース、グラフィックス、ゲームなど、さまざまな用途のものがあります。パソコンを利用するには、これらのパッケージソフトを購入して使用するのが一般的です。

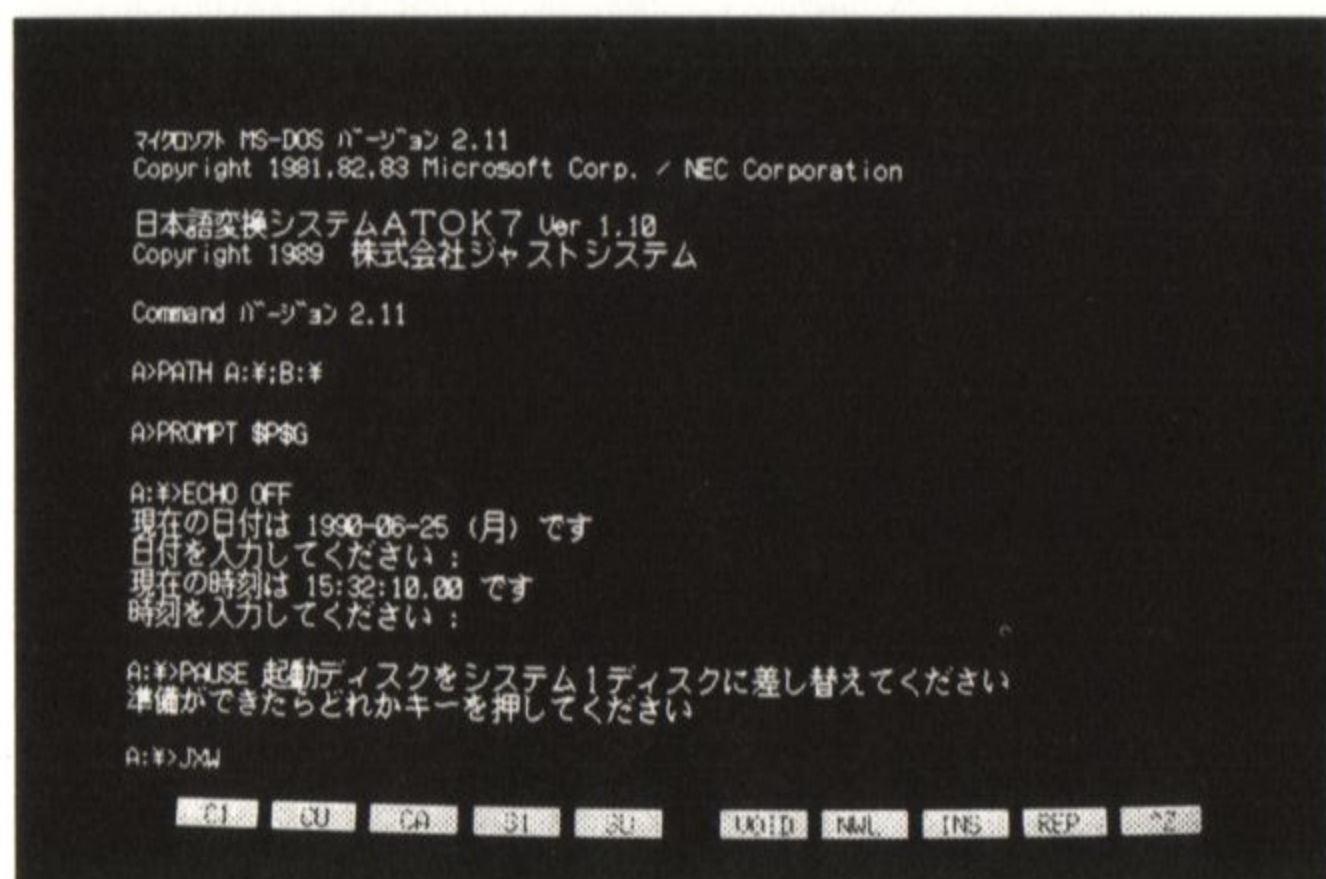
かつて、パソコンを使うということは、イコール、自分でプログラムを組んで、そのとおりパソコンに仕事をさせることでした。それが現在、自分でプログラミングを行う人は、ごくひとにぎりのユーザにとどまっています。では、プログラミング言語はもはや不要なのでしょうか？

いいえ、けっしてそうではありません。市販のパッケージソフトも、やはりプログラムの集まりなのです。書かれたプログラムが、コンピュータにわかる言葉に変換され、すぐに動き出せる状態になっているのが、これらのさまざまなパッケージソフトの正体なのです。プログラムを記述するためのプログラミング言語には多くの種類がありますが、パッケージソフトの大部分のプログラムは、C言語やアセンブラ（19ページ参照）で書かれています。

OSとC言語

6 OSの役割

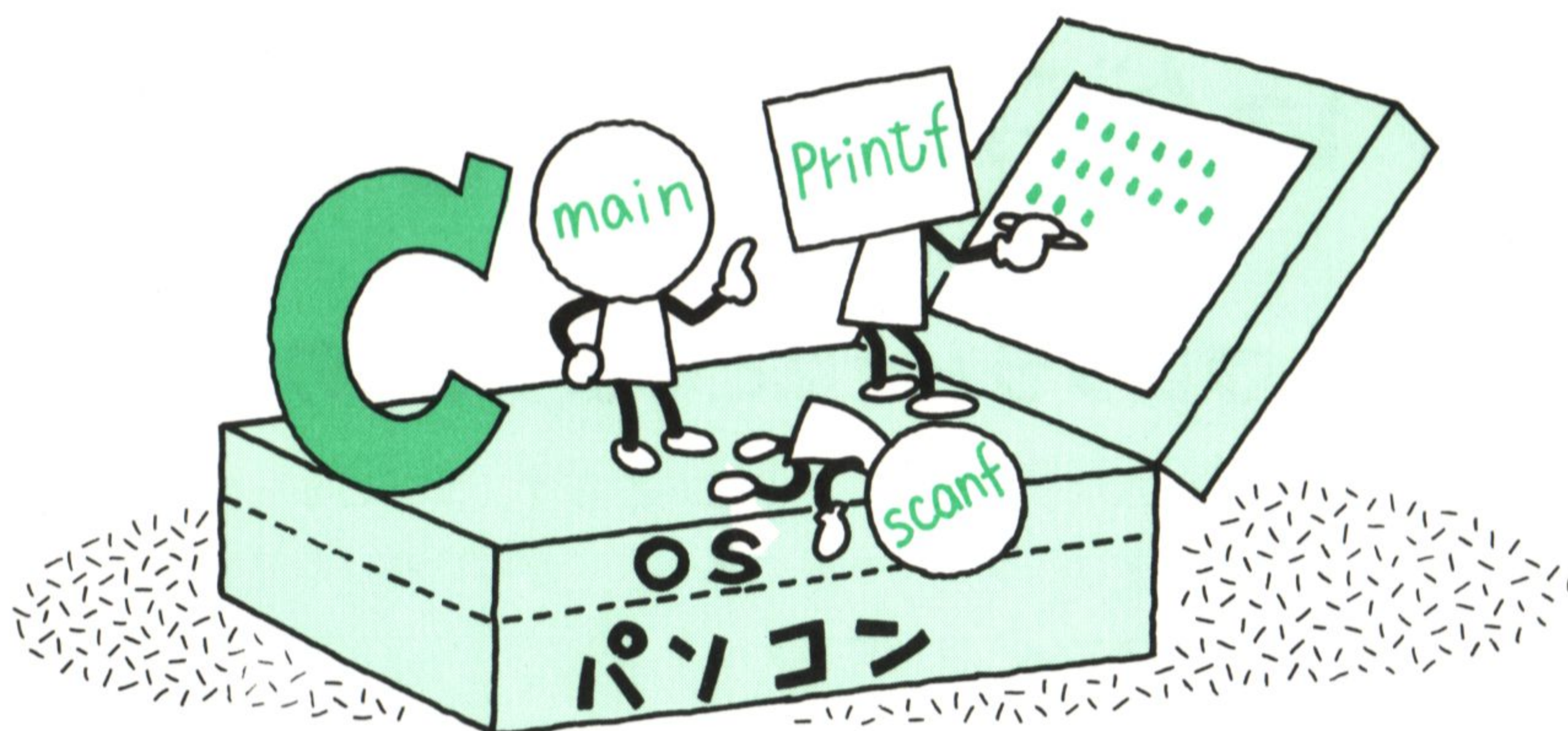
C言語でプログラムを組むためには、OSつまりオペレーティング・システムというものが必要です。OSは、パソコンという体をめぐる血液のようなもので、これがないと、前述したいろいろなパッケージソフトも動きませんし、C言語のプログラムを作ることもできません。そして現在の16ビット・パソコンでは、MS-DOS（エムエス・ドス）というOSが主流になっています。

たとえば「一太郎Ver.4」を起動すると、最初にMS-DOSが起動し、次の「A:¥>JXW」に続いて一太郎の画面が現れます。つまり、まずOSがパソコンに読み込まれ、続いてOSがパソコンとソフトとの間を仲立ちするような形で、目的のソフトを呼び出しているわけです。

▲MS-DOSの起動

▲一太郎のタイトル画面

MS-DOSはMicroSoft-Disk Operating Systemの略で、米国マイクロソフト社の製品です。パソコンが普及しはじめたころは、どのOSが主流になるかについてさまざまな経緯があったのですが、結果的にMS-DOSがトップの地位を占めました。パソコンを起動して、次のような表示（これをプロンプトと言う）が出ているときは、このMS-DOSが動いています（A：¥>、B：¥>、C：¥>　などのような表示の場合も同じ）。

```
A>
```

パソコンのハードウェアにまずMS-DOSをのせ、さらにC言語のコンパイラ(C言語をパソコンに理解させるための道具)というものを起動して、ようやくC言語のプログラムを作る環境が整います。具体的な手順については、PART2でくわしく説明します。

6 C言語が使えるOS、UNIXとMS-DOS

C言語は、もともとUNIX(ユニックス)というOSを開発するために、1972年に作られたプログラミング言語です。UNIXはC言語で書かれ、さらにUNIX上で走るいろいろなソフトもC言語で作られました。しかもプログラムの内容が公開されましたので、大勢の人がC言語のプログラムを共有し、さらに使いやすいように修正されて発展しました。

本来、大型コンピュータ用のソフトウェアは非常に高価なものですが、UNIXは大学などの研究機関へは磁気テープ代程度の値段で提供されたため、世界中にいち早く広まり、日本でも大学などの大型コンピュータ上では早くから使われていました。現在でも、UNIXが搭載されているコンピュータで使われるプログラミング言語はC言語が主流なので、UNIXとC言語は切っても切れない関係にあります。

MS-DOSは、もともとUNIXを手本にして作られたもので、コマンド体系などがUNIXに似ています。ただ、MS-DOSは1人のユーザが1つのソフトを動かすように作られているOSですが、UNIXは複数のユーザが、いくつものソフトを並行して動かすことのできるOSです。このことを、MS-DOSはシングルユーザ／シングルタスク、UNIXはマルチユーザ／マルチタスクと表現しています。

UNIXとともに発展してきたC言語ですから、C言語でプログラミングする際のいろいろなツール(道具)が、UNIXには豊富にそろっています。一方、MS-DOS上で動くC言語コンパイラが安く手にはいるようになりましたので、パソコン上でもC言語を使える環境がしだいに整ってきました。

プログラミングとC言語

プログラムとはどのようなものか

一太郎などパソコンのパッケージソフトにかぎらず、コンピュータを動かすには、ソフトウェア（つまり、プログラムの集まり）がなければなりません。大型コンピュータから、オフコン、ワークステーション、パソコンにいたるまで、コンピュータは「ソフトがなければただの箱」なのです。このソフトを作る（書く）ために使われるのが、プログラミング言語と呼ばれるものです。

プログラミング言語の代表的なものにはBASIC（ベーシック）、COBOL（コボル）、FORTRAN（フォートラン）、アセンブラ、そしてC言語などがあります。

いちばん有名で、簡単なのはBASICです。Beginner's All-purpose Symbolic Instruction Code の頭文字から名づけられたとおり、初心者向けの、どんな目的にも使える簡単な言語です。プログラミング言語とはどういうものかを見るため、BASICの例をあげて説明してみましょう。

◎四則演算のBASICプログラム

```
10 PRINT "四則演算を行います"
20 INPUT "X = ",X
30 INPUT "Y = ",Y
40 INPUT "演算? ",A$
50 IF A$ = "+" THEN PRINT "X + Y = ",X + Y
60 IF A$ = "-" THEN PRINT "X - Y = ",X - Y
70 IF A$ = "*" THEN PRINT "X * Y = ",X * Y
80 IF A$ = "/" THEN PRINT "X / Y = ",X / Y
RUN ↵
四則演算を行います
X = 16 ↵
Y = 5 ↵
演算? + ↵
X + Y =        21
```

（10〜80行：プログラム／RUN以降：実行例）

これを見ると、英語の単語のようなものが並んでいます。それもそのはず、コンピュータは主にアメリカで発達したものですから、プログラミング言語も、彼らの母国語である英語を使って組み立てられています。ですから、日本人より英語を母国語にする人々のほうがプログラムになじみやすいのは、しかたがないことでしょう。

日本語を使ったMIND（マインド）というプログラミング言語もあります。また子供向けのLOGO（ロゴ）という教材用のソフトも、日本語でプログラミングすることができます。日本語でプログラムを書くほうが日本人にはわかりやすいだろうと思われますが、残念ながらコンピュータの専門家の間では、日本語があまり重要視されていません。プログラマのほとんどはプログラミング用の英語に慣れているため、いまさら日本語ではわずらわしい、というのがその理由のようです。

また、BASICをはじめとするプログラミング言語は、すでに世界の共通語として使われているため、英語が得意でなくとも、プログラミングを通して意思を通じることができるという、非常に大きなメリットがあります。

・・・プログラミング言語のいろいろとC言語

プログラミング言語には、実にさまざまな種類があります。有名なものを、別表にまとめておきます。

◉主なプログラミング言語

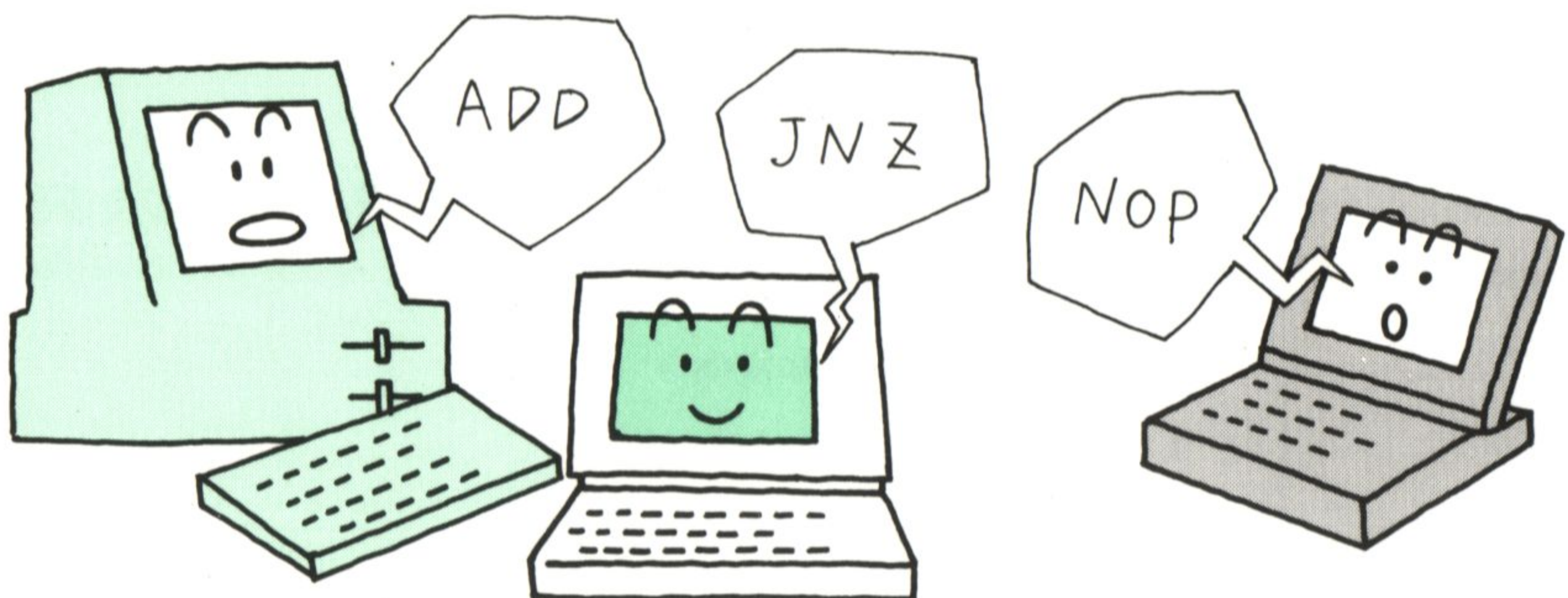
名　称	読みかた	特　徴
BASIC	ベーシック	初心者向きの言語、大きなプログラム開発には向かない
FORTRAN	フォートラン	科学技術計算に適している
COBOL	コボル	事務計算に適している
PL/I	ピーエル・ワン	事務計算に適している
C言語	―――	多用途、アセンブラに似た使いかたができる
PASCAL	パスカル	科学技術計算に適している
PROLOG	プロログ	人工知能向き
LISP	リスプ	人工知能向き
アセンブラ	―――	機械語に最も近い言語

BASICからLISPまでは、「高級言語」と呼ばれるものです。これらは、人間の言葉に近い言語で書き表されます。これに対して、アセンブラは「低級言語」と呼ばれ、機械語に近い言葉で表記されます。高級、低級という呼びかたは、どちらが上等でどちらが下等だという意味で使われているのではありません。単に、機械に近いほうを低級、人間に近いほうを高級と呼び習わしているだけです。

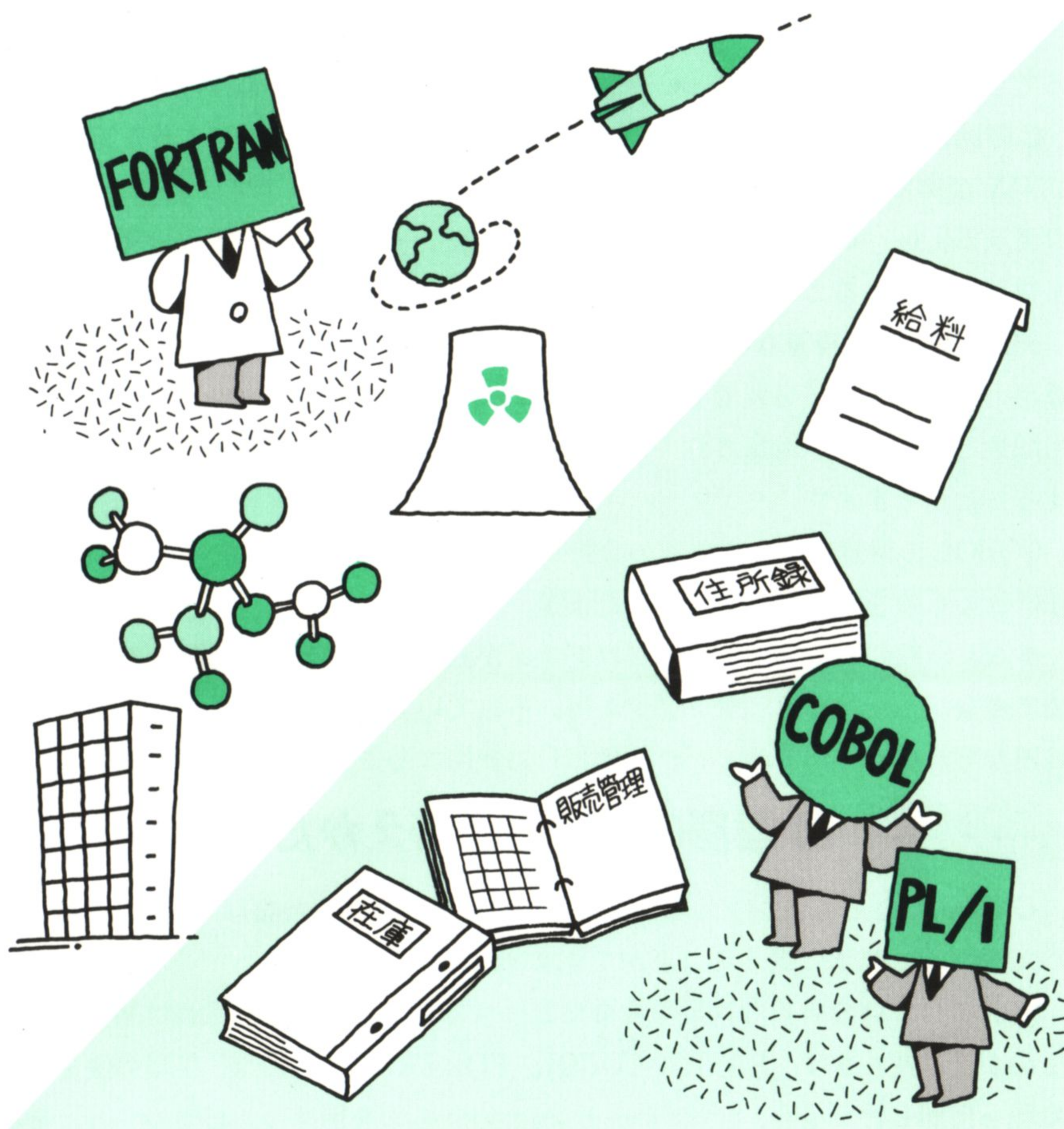
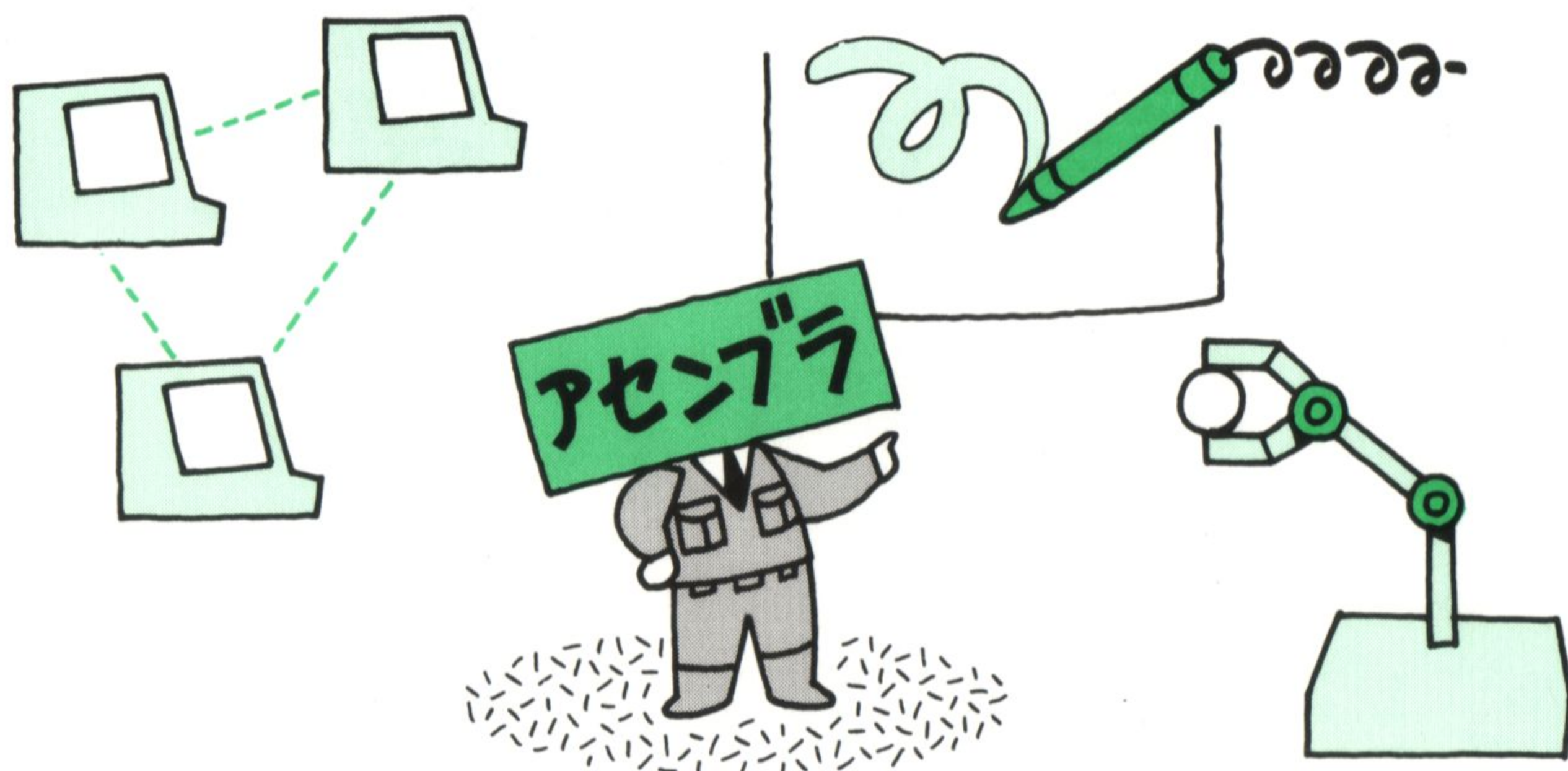
高級言語のうち、FORTRAN、COBOL、PL/I は、主に大型コンピュータ、ワークステーションなどと呼ばれるコンピュータ上で使用されています。パソコン上でプログラムを組む場合、初心者向けにはBASICがあり、実用的なプログラミングにはC言語またはアセンブラが使用されます。

アセンブラは、正しくはアセンブリ言語と呼ぶべきもので、コンピュータが直接理解できる言葉である機械（マシン）語に近い言語です。書かれたプログラムは、ほかの言語に比べて処理速度が速いという長所がある反面、プログラミングに時間がかかることと、使用するコンピュータによってアセンブラが少しずつ違うという障害があります。

これは、同じ日本語でも地方によって方言があるようなものですが、方言の違いはパソコンごとに異なるといっていいくらいで、たとえば日本電気のパソコンと日立のパソコンで使われるアセンブラがまったく違うだけでなく、同じ日本電気のPC-9800シリーズとPC-8800シリーズでも異なります。つまり、PC-9800シリーズで作られたソフトをPC-8800シリーズ上で使おうとしても、スムーズにはいかないのです。こういう点でアセンブラは不便です。

といって、BASICには、処理が遅い、プログラムの構造が複雑になるため共同開発がしにくいなどの欠点があります。もう少し効率のよいプログラミング言語はないか——ということで、C言語が注目されるようになりました。現在では、パソコンのパッケージソフトの大部分がC言語で開発されています。

C言語の特徴と適性

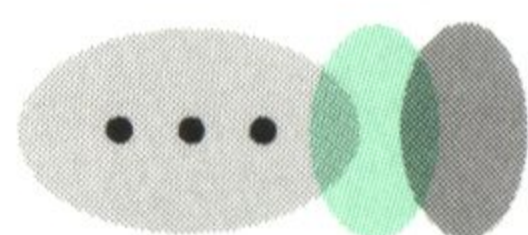
C言語の汎用性

多くのプログラミング言語には、目的によって向き、不向きというものがあります。たとえば、COBOL(コボル)は事務処理に向くとか、FORTRAN(フォートラン)は数値計算に向くとか、アセンブラはハードウェアを直接操作するのに向くなど、用途に応じて言語が使い分けられていました。

では、C 言語はどのような仕事に向く言語なのでしょうか。これは、ひと言でいってしまうことができます。C 言語は、どんな仕事にも向く言語なのです。

C 言語は、前述したようにUNIX（ユニックス）という OS を作るために開発された言語です。UNIX が作られると、UNIX 上で使ういろいろなソフト（アプリケーション・プログラム）も、すべてC 言語で開発されました。その結果、C 言語は実にさまざまな用途に使えることがわかり、現在では多くの分野に力を発揮しています。

FORTRAN の得意な科学計算の分野を処理するためには、いろいろな科学計算用のプログラムが必要です。C 言語には、それに対応するものとして、数値計算用関数というものが用意されています。また、足りない関数などは、自分で作成することができます。

COBOL の得意なデータ処理の分野に対しては、C 言語の構造体やポインタと呼ばれるものを利用することができます。

また、一般の高級言語では、プログラムからハードウェアを制御することが直接できなかったため、アセンブラが用いられていました。C 言語ならこのような処理も可能なので、アセンブラの代わりに使用することもできます。

C 言語のしくみと考えかた

C 言語は、とてもシンプルな言語です。プログラムの命令語は、ほかの言語に比べて少なく、簡単に覚えることができます。

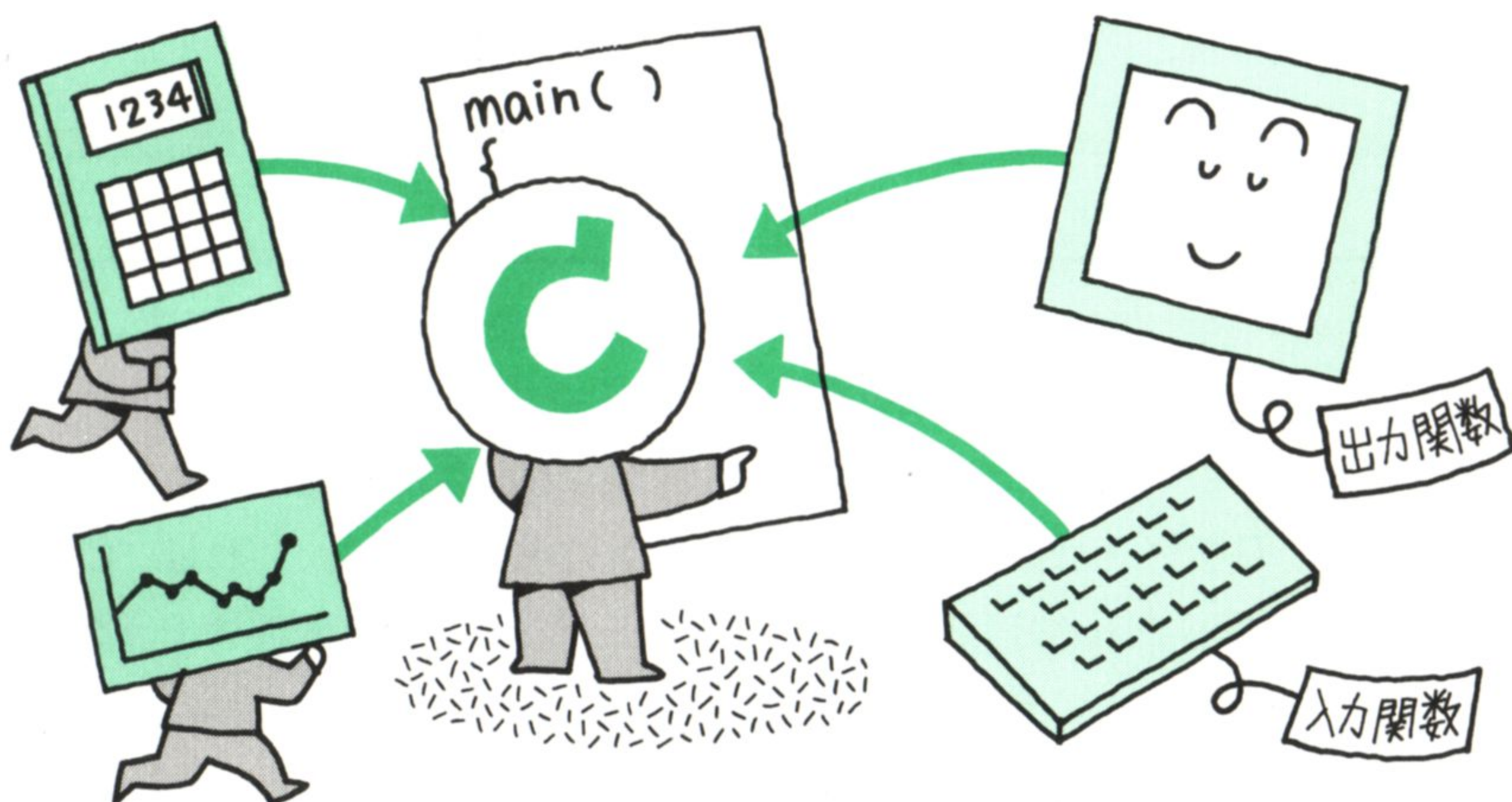
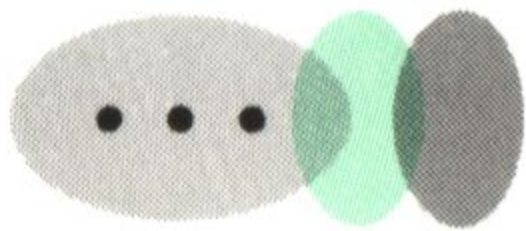
それでは、いろいろな複雑な命令はどうするのかというと、それは関数というものが受け持ちます。BASIC（ベーシック）やCOBOL（コボル）、FORTRAN（フォートラン）などでは、手順の決まった処理を何回も行う場合、「サブルーチン」というものを用意し、必要のつど、それ

を呼び出して処理させます。サブルーチンは、いわばプログラムの部品にあたるものです。それに対応するのがＣ言語の関数です。

Ｃ言語では、画面への出力、キーボードからの入力、プリンタへの出力、ファイルへの入出力などの標準関数が用意されていて、それぞれの役目を果たします。そのほか、前述したように数値計算用の関数や、メモリ割り当て関数、グラフィック関数などもあります。

関数は、自分で新たなものを作ることもできます。さらに、あらかじめ用意されている関数を使わずに自分で作成したものを使うこともでき、上級プログラマのなかには、ほとんどの標準関数を自作している人さえいます。

また、アセンブラの代わりになると述べたように、Ｃ言語ではハードウェアの動きを制御することもできます。それでいて、アセンブラとは異なり、機種への依存度（マシン依存性）が低いので、異なるコンピュータでもほとんど同じプログラムを走らせることができます。こういった働きのために、Ｃ言語はプログラマたちの間で愛用されるようになりました。

BASICにはいろいろ便利なコマンドが用意されているので、コマンドを覚えると簡単に使うことができます。単語を覚えるだけなので、BASICはいわば暗記型の言語だといえます。一方Ｃ言語は、覚える言葉が少ないかわりに、自分で関数を作っていかなければならないので、定義型の言語だということができるでしょう。またＣ言語では、関数に名前をつけるという作業が必要です。関数名、変数名、構造体名などなど、いろいろ名前をつける機会の多いのがＣ言語の大きな特徴です。

C 言語と BASIC

簡単なプログラムを書くとき、C 言語でもBASIC（ベーシック）に似た単語が使われます。C 言語と BASIC の命令語のいくつかを、一覧表にして比べてみましょう。BASIC の命令語はコマンドと呼びます。それに対して、C 言語の命令語はステートメント（文）と呼ばれます。C 言語のステートメントは10個程度しかなく、主にいろいろな宣言を行ったり、プログラムの流れをつかさどる働きをします。

◉ C 言語と BASIC の命令の対比

C 言語	BASIC	意　味

for (i = 0; i < 10; i++) {
　　……
　}
	FOR I=1 TO 10
	　　……
	NEXT I
		指定された回数だけその作業をくり返す

while (x != 0)　{
　　……
　　}
	WHILE X<>0
	　　……
	WEND
		ある条件のあいだ、その作業をくり返す

if (a > 10)　{
　　……
　　}
	IF A>10 THEN
	……
	……
		条件によって行う作業を変える

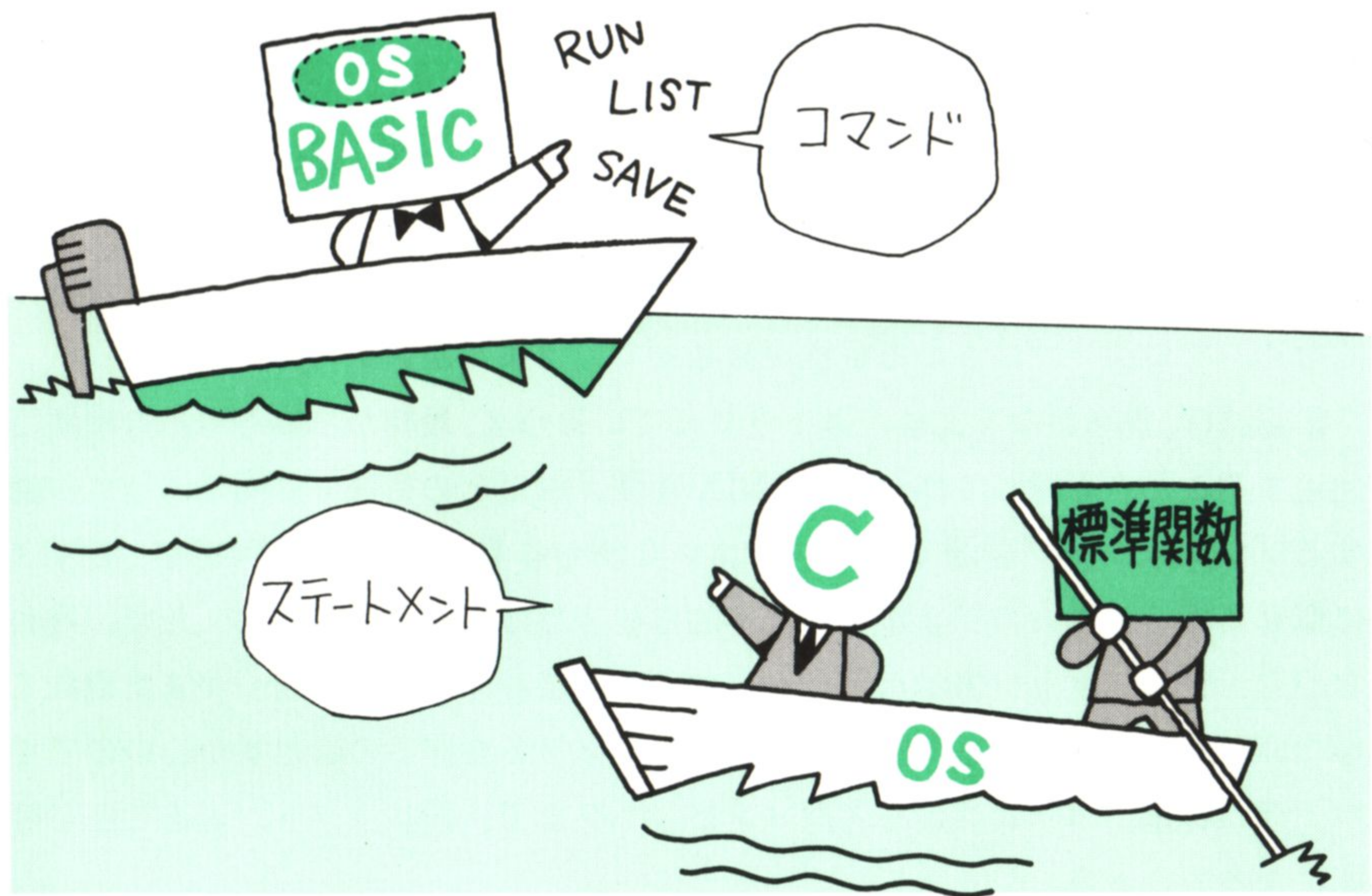
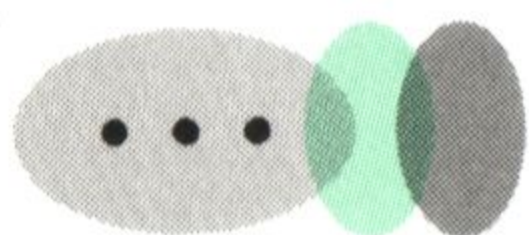
printf("a = %d", a);	PRINT "a = ";A	画面に文字を表示する

BASIC のコマンドが、すべて C 言語に対応するわけではありません。BASIC と C 言語とを対応させると、命令を次表の 3 種類に分けることができます。

BASIC ではキーボードからの入力や画面への表示などの働きをするのもコマンドですが、C 言語では標準関数と呼ばれるものを使います。これは“C 言語の体系の外にあるもの”と呼ばれています。ユーザ自身が標準関数のようなものを作ることもできるからです。BASIC の場合は、自分でコマンドを作ることができません。

プログラムを実行したりファイルを作ったりするコマンドも、BASIC にはありますが、C 言語にはありません。そのかわり OS の命令を使います。この性質は C 言語だけでなく、FORTRAN、COBOL、PASCALなどの言語でも同じです。OS の命令までも含む BASIC のほうが、特殊な言語なのです。

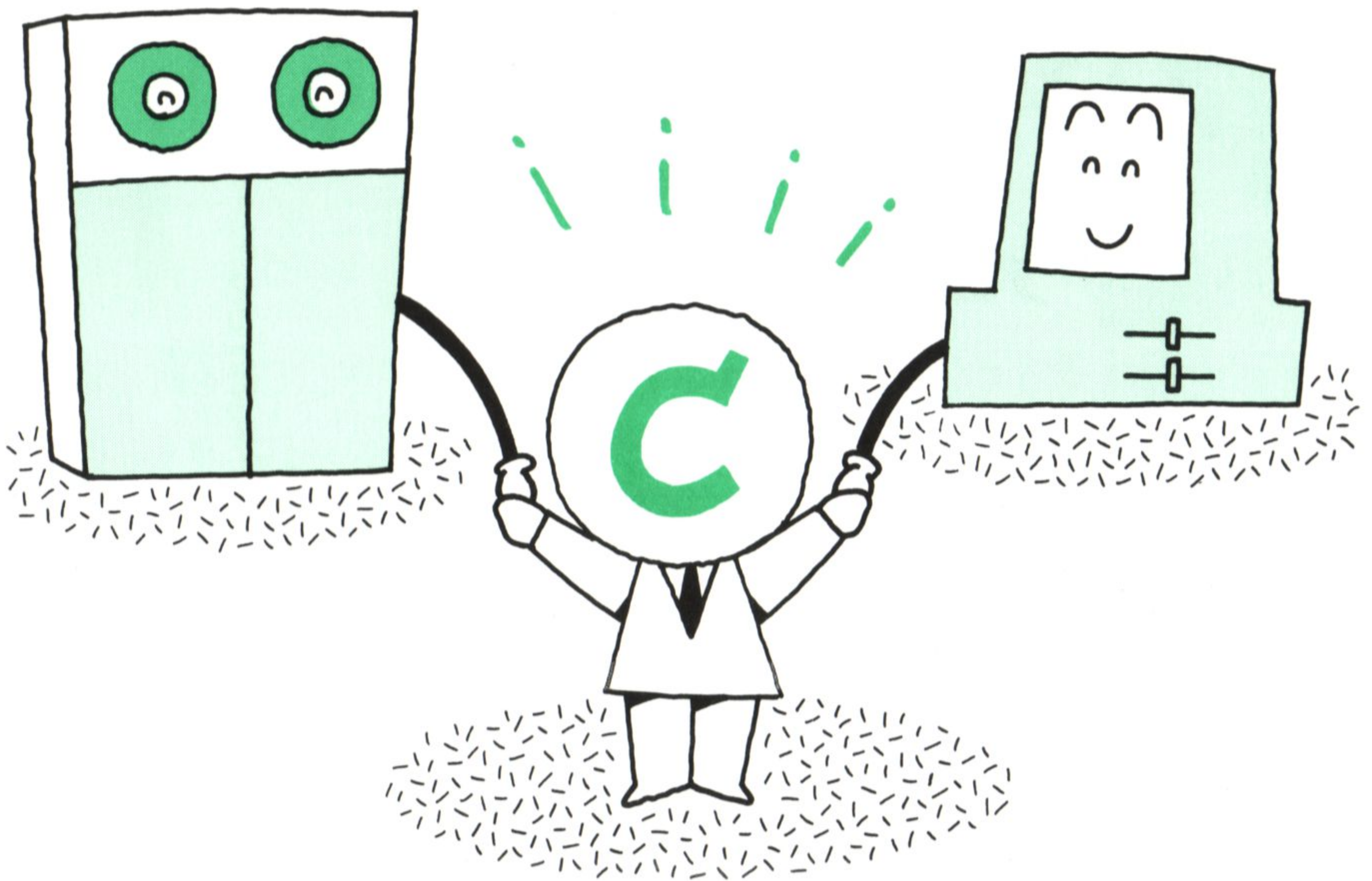
◉C言語と BASIC の命令の分類

分　類	BASIC のコマンド	C言語
プログラムの流れを制御する命令	FOR～NEXT WHILE～WEND IF～THEN	ステートメント for、while、if
入力や出力に関する命令	PRINT INPUT	標準関数 printf( )、scanf( )
パソコンに直接与える命令	RUN LIST FILES LOAD	C言語にはない OS の命令で行う

C 言語の移植性

C 言語は、機種によらず同じプログラムを走らせることができるという大きな特徴があります。これはコンピュータの歴史のなかでは実に画期的なことです。かつては、コンピュータが違うとそこに使われるプログラミング言語も異なっていました。A 社の大型コンピュータで作ったプログラムを B 社のコンピュータに乗せかえるだけで、何か月もかかるというような事態がざらにあったのです。異なるコンピュータ上で同じプログラムを走らせるというのは、むずかしい問題をはらんでおり、現在でも完全に解決したわけではありません。これを解決する手がかりがUNIX(ユニックス)であり、C 言語であるといえます。

現在、コンピュータのハードウェアは、メーカ、型式（大型か中型か小型かなど）、CPU（中央処理装置）の種類などによって、細かく異なります。パソコンの世界をながめてみると、アメリカではIBM PC 機とその互換機が主流を占めていて、アップル社のマッキントッシュも有力なシェアを持っています。一方、日本の場合、IBM とは互換性の少ない日本電気の PC-9800 シリーズが主流となっています。日本で IBM PC 機と互換性があるのは、各社から出されている AX パソコンと呼ばれるものです。互換性というのは 1 つのソフトを何の変更もなくどのパソコンにも乗せるということですが、現状ではまだまだ無理です。

パソコンのソフトを見るとわかりますが、同じソフトでも、PC-9800 用、あるいは東芝 J-3100 用などと、機種に応じた別々の製品が売られています。これも、別の機種では同じソフトが動かないためです。もし、機種に関係なく動くようになれば、1 本のソフトを変更することなく、さまざまなパソコンのユーザに購入してもらうことができます。こういう時代になると、ユーザにとって便利なだけでなく、プログラマにかかる負担が激減するのはまちがいありません。

C 言語は、基準となる文法がはっきりしているうえ、最初からいろいろな機種で使用することを考慮して作られ、UNIX が使えればどの機種でも同じように作動するプログラミング言語です。パソコンで使える C 言語もMS-DOS(エムエス・ドス)も、UNIX に似せて作られたものですから、基本的な仕様はほとんど同じです。ただ、残念ながら、グラフィック表示などハードウェアに依存する部分の統一がまだ取れていないため、C 言語でも完全に互換性のあるソフトを作るのはむずかしいのですが、近い将来、パソコンから大型コンピュータまで、同じソフトが走ることも可能になるでしょう。

現在、パソコンの世界は、16ビット機から32ビット機への転換期に入っています。1988年、日本電気からPC-9800RAが約50万円で売り出されたのを皮切りに、ぞくぞくと32ビット・パソコンが登場してきました。

いまの16ビット・パソコン上で動いている代表的なソフトは、ほとんど640KBのメモリをいっぱいに使用していますから、たとえば2種類のソフトを並行して走らせたり、もっと細かく画像処理をしたい場合など、現状のハードウェアではまかないきれません。そこで、32ビット機に期待がかけられているのです。

32ビット機の時代になると、パソコンが大型機とほとんど同じ機能を持つことができるといわれています。それにつれて、いままでパソコンはパソコン、大型は大型と、それぞれ独自のソフトウェアが走っていたのが、共通のOS上で、共通のソフトを使えるようになっていきます。そのための共通の言語として、C言語が使われるだろうと予想されているのです。

現在のC言語ブームは、こうしてみると、次にくる32ビット機時代への先駆けだといえるかもしれません。16ビット・パソコン上では主要なソフトが出つくした感があり、次の32ビット時代のOSといわれるOS/2（オーエス・ツー）やUNIXに関心が集まっていますし、それらのもとで使われるプログラミング言語、C言語への期待が高まってきています。

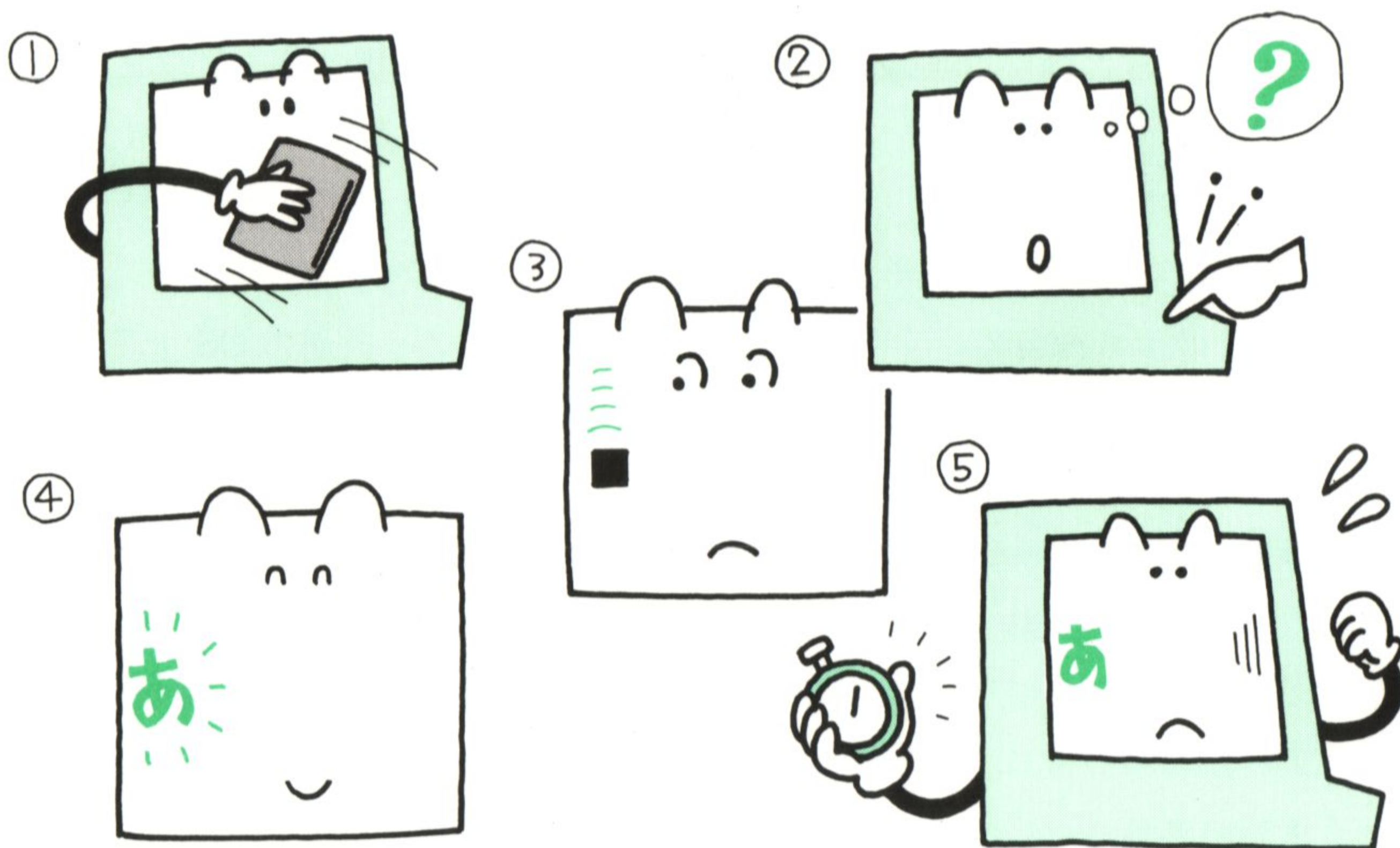
プログラミングのあらまし

プログラムの働き

プログラムというのは、どんな働きをするのでしょうか？　これを知っていると、コンピュータそのものの働きやしくみが実感できるようになります。たとえば、「ある文字をキーボードから入力して画面に表示させるプログラム」を考えてみましょう。１文字を表示させるくらい簡単なようですが、実際にこれをコンピュータに実行させるには、次のような手順が必要です。

①画面上の不要な文字などを消す（画面をきれいにする）。
②キーボードから入力された文字を認識する。
③文字を表示する場所にカーソルを移動する。
④その文字を表示する。
⑤そのまま何秒間か同じ状態を続ける（でないと、表示が一瞬で消えてしまう）。

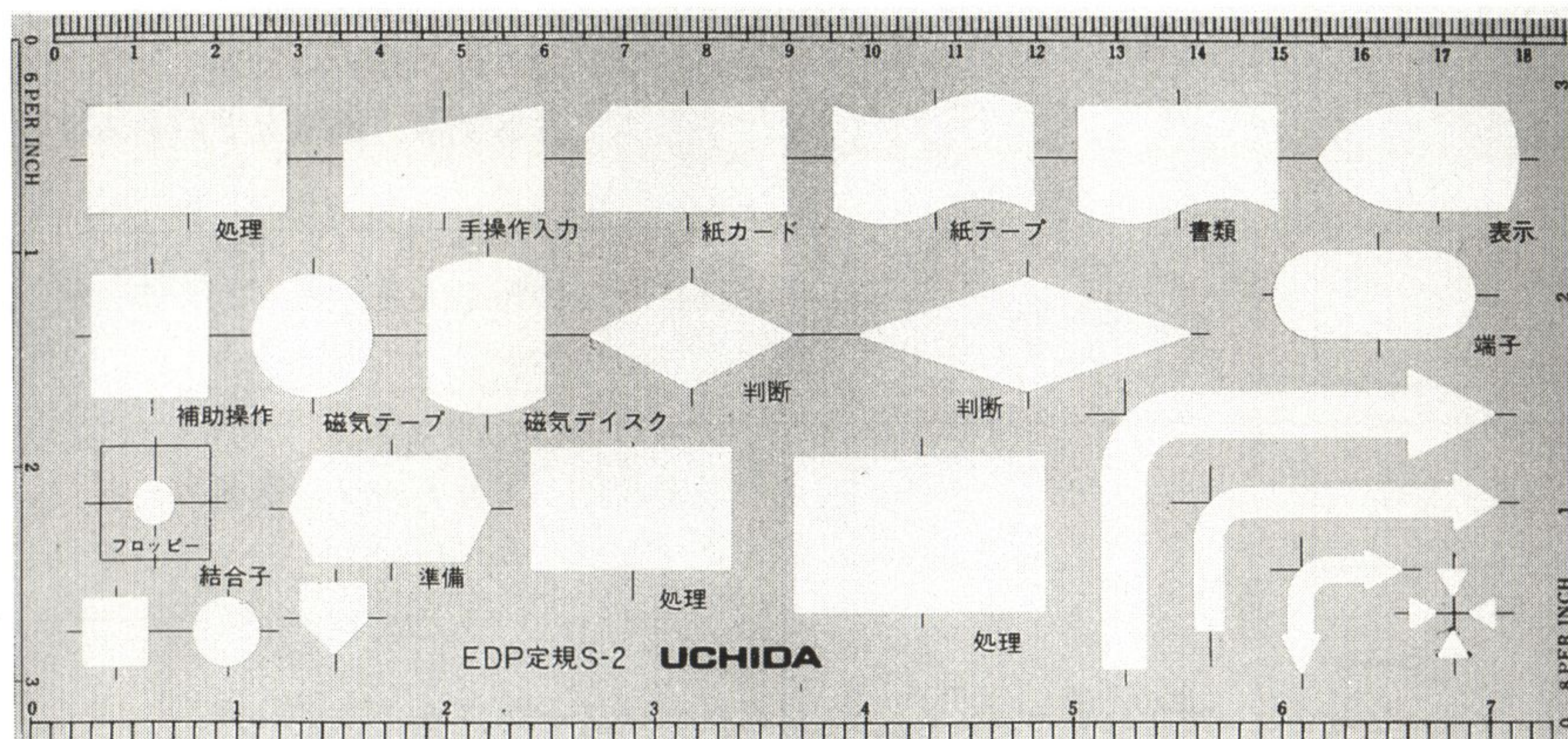
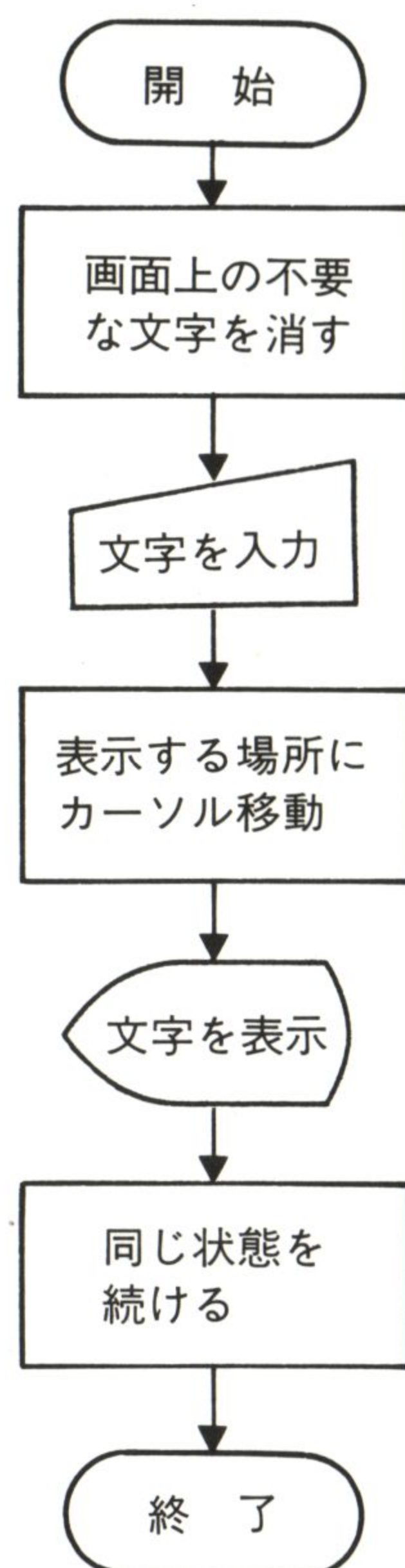
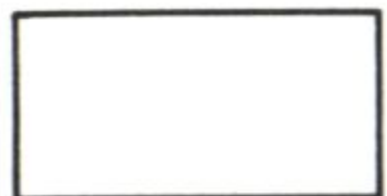
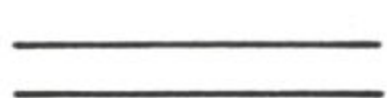
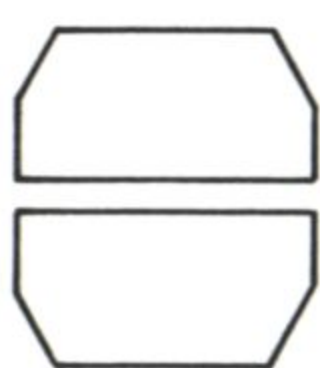
言葉でいうと簡単ですが、これをプログラムにするには、実に何段階にも分けて命令しなければなりません。このように、コンピュータにわかるように命令を並べる、というのがプログラミングの基本です

①　②　③　④　⑤

フローチャート

プログラミングを行う際には、最初にフローチャートと呼ばれるものを書きます。プログラミングの手順を各仕事単位に図で表したもので、日本語では流れ図といいます。前例の「ある文字をキーボードから入力して画面に表示させるプログラム」をフローチャートにすると、右のようになります。

フローチャートは、プログラムの手順を自分で構築するために書くものです。目的の仕事を漠然と理解するだけでなく、それをプログラムに適したものにしなければなりません。簡単なプログラムの場合はフローチャートを省略するプログラマもいますが、最初のうちは、どのようなプログラムにするかをしっかり把握するためにも、フローチャートを必ず書くようにしてください。

フローチャートには、JIS によって定められた各種の記号（次ページ参照）が使われます。このうち、データ記号の入力や表示など入出力関係、処理記号の処理や判断などが特に重要です。なお、フローチャート用のテンプレート（型定規。写真参照）が文具店などで販売されています。

開　始
画面上の不要な文字を消す
文字を入力
表示する場所にカーソル移動
文字を表示
同じ状態を続ける
終　了

▲フローチャート用のテンプレート

◉主な JIS フローチャート記号一覧

1 データ記号

【資料】JIS ハンドブック　情報処理(日本規格協会編)

記号	名称	意味
	データ	媒体を指定しないデータ
	記憶データ	処理に適した形で記憶されているデータ。媒体は指定しない
	内部記憶	内部記憶を媒体とするデータ
	順次アクセス記憶	順次アクセスだけ可能なデータ。媒体としては磁気テープ、カートリッジテープ、カセットテープなどがある
	直接アクセス記憶	直接アクセスが可能なデータ。媒体としては磁気ディスク、磁気ドラム、フレキシブルディスクなどがある
	書類	人間の読める媒体上のデータ。媒体としては印字出力、光学的文字読み取り装置または磁気インク読み取り装置の書類、マイクロフィルム、計算記録、帳票などがある
	手操作入力	手で操作して情報を入力する、あらゆる種類の媒体上のデータ
	カード	カードを媒体とするデータ。磁気カード、マーク読み取りカードなど
	せん孔テープ	せん孔テープを媒体とするデータ

	表示	人が利用する情報を表示する、あらゆる種類の媒体上のデータ。表示装置の画面、オンラインインディケータなど

❷処理記号

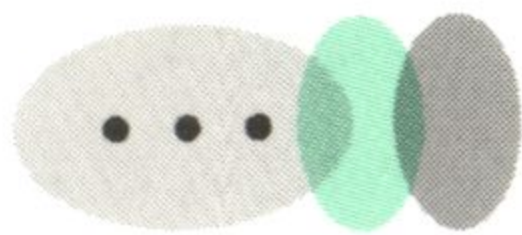
記　号	名　称	意　味
	処理	任意の種類の処理機能。たとえば情報の量、形、位置を変えるような定義もしくは演算群の実行、または次に続くいくつかの流れの方向の１つを決定する演算もしくは演算群の実行
	定義ずみ処理	サブルーチンやモジュールなど、別々の場所で定義された１つ以上の演算または命令からなる処理
	手作業	人手による任意の処理
	準備	その後の動作に影響を与えるための命令、または命令群の修飾。スイッチの設定、指標レジスタの変更、ルーチンの初期設定など
	判断	１つの入り口といくつかの択一的な出口を持ち、記号中に定義された条件の評価に従って唯一の出口を選ぶ判断機能またはスイッチ形の機能
	並列処理	２つ以上の並行した処理を、同期させること
	ループ端	２つの部分からなり、ループの始まりと終わりを表す。テスト命令の位置に応じて、ループの始端または終端の記号中に、初期値、増分(ぞうぶん)、終了条件を併記する

コーディング

大型コンピュータなどでプログラムを作る場合には、フローチャート作成の次に、プログラムを紙に書くという工程が必ず入ります。この作業を、コーディングと呼びます。プログラミング言語によっては専用のコーディング用紙が用意されていて、それが使用されます。パソコンでプログラミングを行うときは、直接パソコンに向かってプログラムを作っていく人がほとんどですが、慣れないうちはまずフローチャートを書き、コーディングしていったほうが、まちがいのないプログラムを作ることができます。

```
SEQUENCE  A   B              COBOL    STATEMENT
010010   ***** ショキ  セッテイ  ******
   020       MOVE     ZERO      TO     TOR-RECORD
   030                                 SHO-RECORD
   040                                 REV-NO
   050                                 BOX5-DATE  GAMEN-BLINK-CONTROL.
   060       MOVE     1         TO     IN-BOX1    IN-BOX2.
   070       MOVE     010100    TO     GA-N-DENKU.
   080       MOVE     SPACE     TO     GA-N-TORNO
   090                                 GA-N-SHONO
   100                                 GA-N-SHUKKABI.
   110       PERFORM  SCREEN-CLEAR.
   120       REKFORM  CLEAR-PROC.
   130       MOVE     GA-N-HEAD        TO     OLD-HEAD.
   140       MOVE     GA-N-HEAD        TO     OLD-WK-IN.
   150       MOVE     ZERO             TO     OLD-SW-OKURI-NO.
   160   ***** カンリ  ヒズケ  *******
   170       PERFORM  KANRI10-READ.
   180       IF       SW-INV    =      1
```

▲コーディング用紙（COBOL 用）

①フローチャートを書く　②用紙にプログラムを書く　③キーボードから入力

プログラムの作成

コーディングまで終わったら、プログラミング言語を使って、いよいよプログラムを入力していきます。C言語で「ある文字をキーボードから入力して画面に表示させるプログラム」を書くと、次のようになります。

```
F ファイル  E 編集  V 表示  S 検索  R 実行  D デバッグ  C 関数          H ヘルプ(F1)
                              B:¥hyouji.c
/* ある文字をキーボードから入力して画面に表示させるプログラム  */
#include <stdio.h>

/* 画面全体を消去する */
cls()
{
        printf("¥x1B[2J");
}

/* 指定の行、桁にカーソルを移動する
        行の範囲  0～24まで
                桁の範囲  0～79まで
    範囲外の値はリターンコード-1を返す      */
int cusol(a, b)
char a, b;
{
        if ( 0 > a || a > 24 ) return -1;
        if ( 0 > b || b > 79 ) return -1;
        printf("¥x1B[%2d;%2dH", a, b);
        return 0;
}
プログラムリスト: <なし>    動作状況: <コンパイル完了:実行できます>    00021:002
                              B:¥hyouji.c
/* 同じ状態を続けるためのカウント */
count()
{
        unsigned long i;

        for ( i = 0; i < 200000L; i++) ;
}

/* 文字Aを画面の中央に表示します */
main()
{
        char a;

        cls();                    /* 画面上の不要な文字を消す */
        a = getch();              /* 文字を入力 */
        cusol(10,40);             /* 画面の中央へカーソル移動 */
        printf("%c", a);          /* 文字を表示 */
        count();                  /* 同じ状態を続ける */
}
プログラムリスト: <なし>    動作状況: <コンパイル完了:実行できます>    00041:002
```

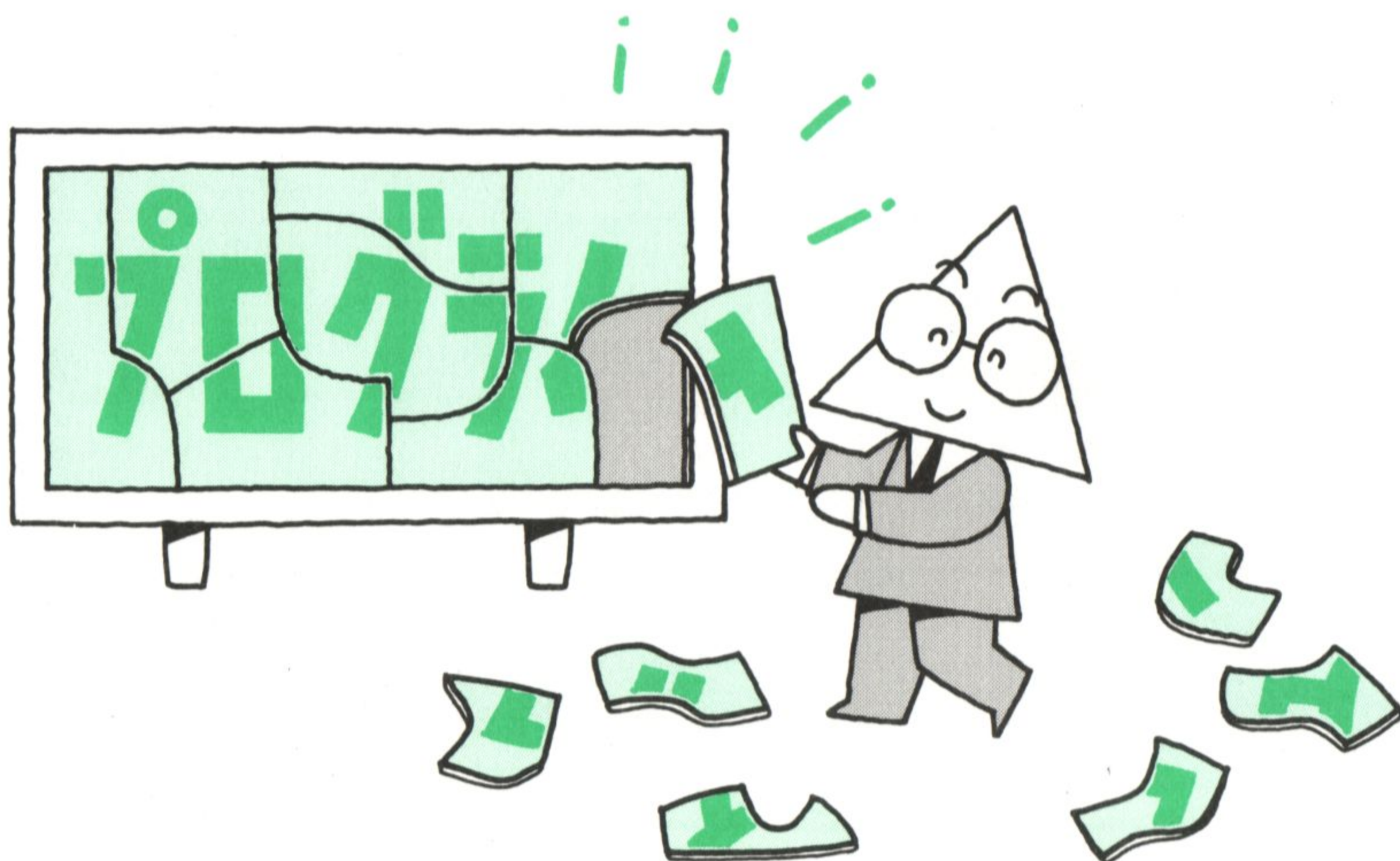
▲C言語のプログラム（Quick Cの画面）

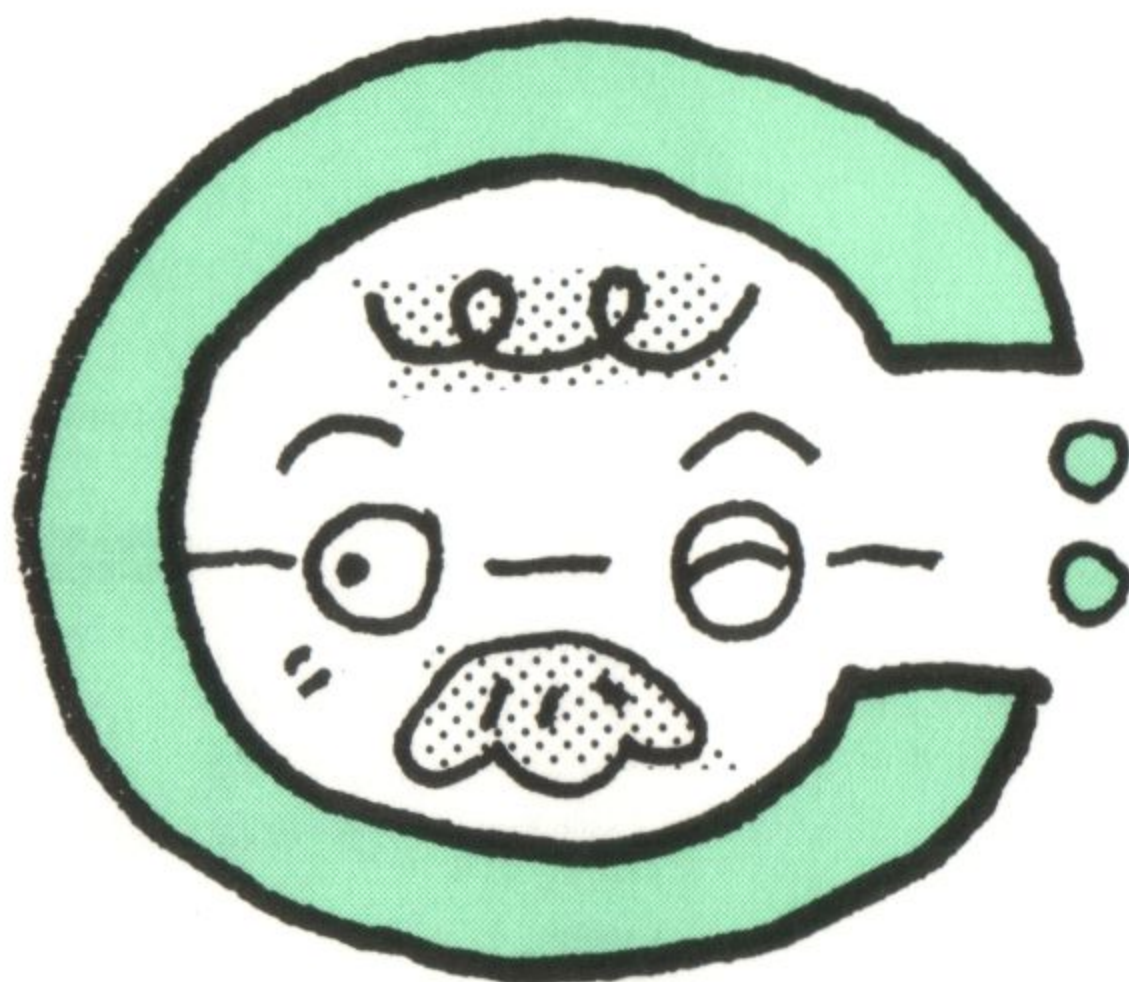
こうしてプログラムができたら実際に動かすわけですが、もう少し細かい、具体的な手順についてはPART 2で説明します。いまは、プログラムとはこんなものだということを、なんとなくでもよいですから理解してください。

C 言語のプログラムでは、流れや考えかたをわかりやすくするため、1つひとつの部品をあまり大きくせず、適当な大きさにまとめることがたいせつです。1つのプログラムはだいたい50～60行以内にするのが理想的。例にあげたプログラムでわかるように、C 言語のプログラムは空白が多く、すっきりした感じにまとまっています。

1つの部品をあまり大きくせず、部品をたくさん作って組みあげていくというプログラミングの手法を、構造化プログラミングといいます。ちょうど、いろいろなパーツを組み合わせて模型飛行機を作るように、プログラムもいくつもの部品を組み合わせて1つの完成品となります。C 言語は構造化プログラミングに適した言語です。

一方、BASIC（ベーシック）はこの構造化には不向きの言語です。構造化ができないため、大きなプログラムが作りにくいのです。また、1つのプログラムが最初から終わりまでつながっているので、一部を別のプログラムで利用したいと思ってもうまくいかず、もう一度書きなおすほうが早かったりします。その点、C 言語は部品だけを取り出しても利用できるので、自分用の部品をたくさん蓄えておけば、プログラミングがますます簡単になります。ほかの人が利用したいときでも、使いかたさえ知っていれば、部品の内容を知らなくてもよいので非常に便利です。

PART 2

C言語プログラミングの基礎知識

田川部長、いよいよC言語に触れる。

田川　やあ、またきたよ。パソコンなどC言語を取り巻く周辺の事情はわかったけど、C言語そのものの使いかたをもっと知りたいね。
玲子　今回は、C言語のプログラムを作るために必要な道具と、実際のプログラム作りの手順といったような話をしましょう。プログラムを作るのに必要なものはなにかしら。
田川　パソコンだろう、それから、エーと……あ、そうそう、MS-DOS(エムエス・ドス)も。
玲子　いちばん大事なのは、C言語のソフトね。C言語のソフトはC言語でプログラムを作るためのものよ。「Quick C」や、「Turbo C」などの名前を聞いたことはないかしら？　こういったソフトをまとめてC言語のコンパイラと呼びます。
田川　そういえば、書店のC言語のコーナーにそういうタイトルのが置いてあったようだ。
玲子　C言語のインタプリタもあるのよ。なかでは「RUN/C」が代表的。
田川　コンパイラとかインタプリタというのが、まだもうひとつわからないな。
玲子　C言語はそのままではコンピュータに理解できない高級言語なので、コンピュータにわかる機械語に変えなければならないのよ。その方法に2通りあって、コンパイラ方式とインタプリタ方式に分かれているわけ。
　たとえばBASIC(ベーシック)はインタプリタ方式で、通訳のようにプログラムを1行機械語に直しては1行実行するという方法でプログラムを実行していく。一方コンパイラは、プログラムをまとめて機械語に翻訳してから一度に実行するというやりかたなの。C言語のほとんどはコンパイラだけど、BASICに似た使いかたができるように作られたインタプリタが「RUN/C」なのよ。
田川　それじゃ、コンパイラやインタプリタで、いったいどのようにプログラムを作るのか、じっくり聞かせてもらうことにしよう。

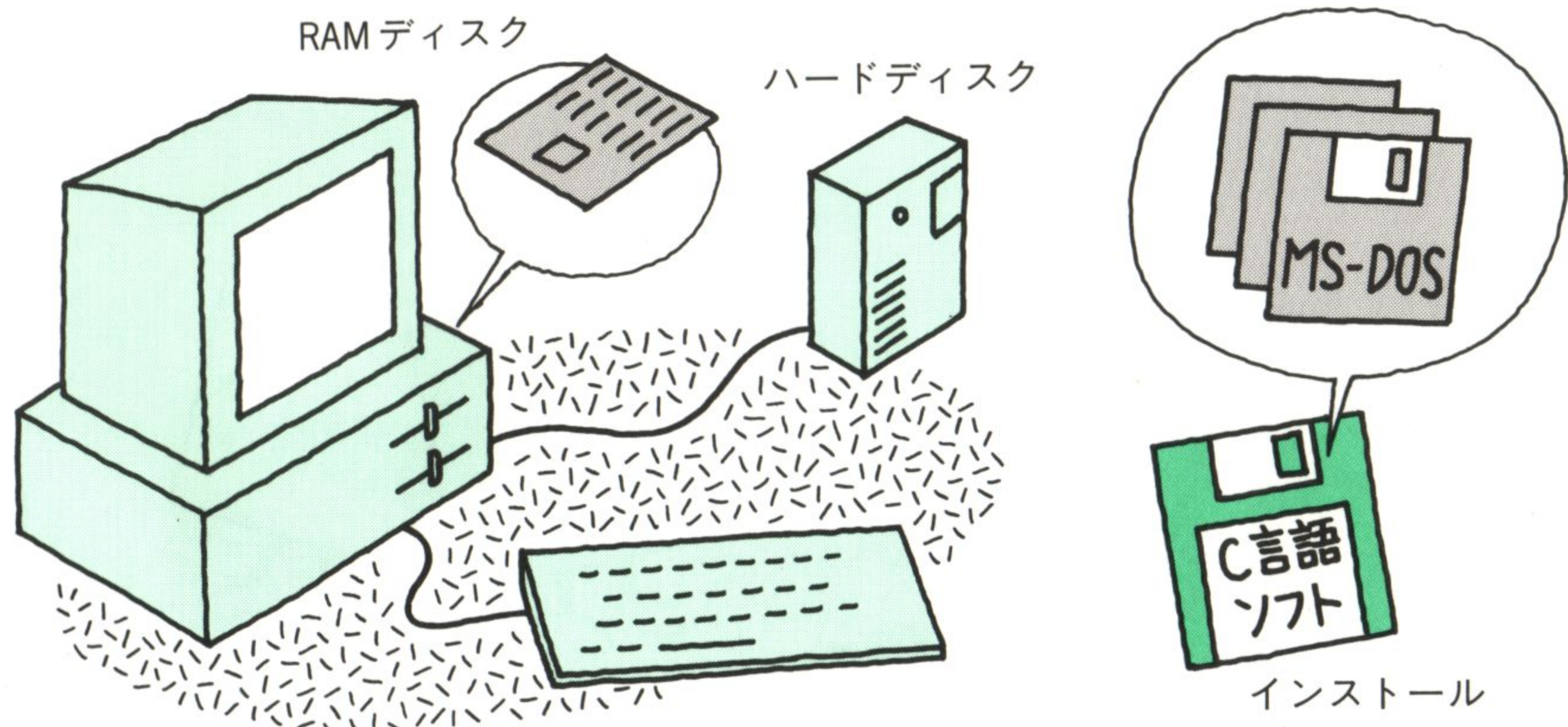
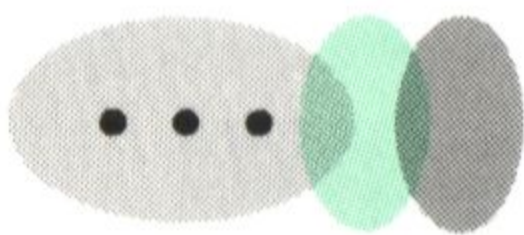
C言語を使うための準備

必要なハードウェアとOS

C 言語は、大型コンピュータからパソコンまで、いろいろなコンピュータ上で動かすことができますが、本書ではパソコンのうち最も代表的な NEC の PC-9800 シリーズを使って話を進めていきます。その他のパソコンでも、MS-DOS（エムエス・ドス）が使える機種なら、同じように操作することができます。

とりあえずパソコンの本体とキーボード、ディスクドライブがあればハードウェアはOK！ RAM（ラム）ディスクかハードディスクがあるとより快適なハードウェア環境となりますから、あればそれらも接続してください。必要なメモリ容量などはC 言語のソフトによって異なるので、次ページの表を参照してください。

C 言語のコンパイラやインタプリタを使うには、MS-DOSが必要です。これは、あらかじめC 言語のソフトのほうへ組み込んで（インストール）おきましょう。ソフトのマニュアルにはMS-DOS の組み込みかたが説明されているはずですから、手順よく操作しておきます。インストール用のプログラムを持っているコンパイラなら、それを実行するだけでMS-DOS を組み込んでくれます。

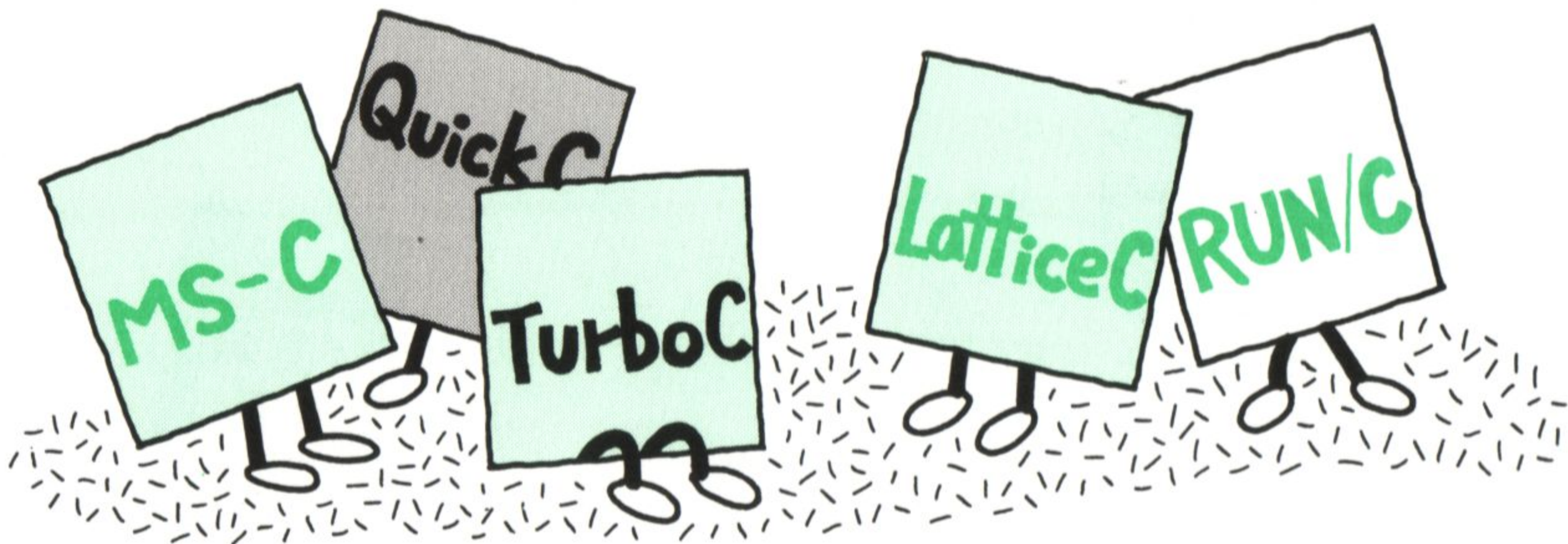
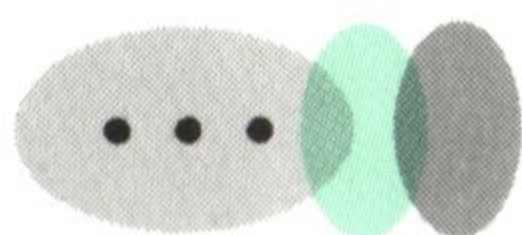
また、パソコンには、OS としてUNIX（ユニックス）を使うものもあります。何度か述べたようにC 言語はUNIX とともに発達してきた言語ですから、UNIX が使用できるコンピュータでも、C 言語を使用することができます。

C言語ソフトの種類

現在発売されているC言語のソフトの主なものは、別表のとおりです。価格は、安価なもので1万円弱、高価なものになると10万円近くします。

◉主なC言語ソフト　1990年5月1日現在

製品名	開発（米国）	発売（日本）	メモリ(KB)	種類	価格
MS-C	Microsoft	マイクロソフトジャパン	640	コンパイラ	98,000円
Lattice C	Lifeboat	ライフボート	640	コンパイラ	98,000円
Quick C	Microsoft	マイクロソフトジャパン	640	コンパイラ 統合開発環境	20,000円
Turbo C	Boland	ボーランドジャパン	640	コンパイラ 統合開発環境	29,800円
Power C	MIX SOFTWARE	システム・ワン	384	コンパイラ	9,801円
Advanced RUN/C	Lifeboat	ライフボート	512	インタプリタ	29,800円

「MS-C」はMS-DOSを開発したマイクロソフト社の製品で、MS-DOSとの相性がいちばんいいようです。「Quick C」と「Turbo C」は低価格のうえ、画面上でメニューを見ながら操作できるという、初心者向きに作られたコンパイラです。「Lattice C」はいろいろなメーカのコンピュータ上で使われているので、他機種への乗せかえがしやすいコンパイラです。インタプリタ形式の「RUN/C」は、前述したようにBASIC（ベーシック）と同様の命令で動かすことができるので、BASICに慣れた人に使いやすいことと、操作が簡単なことが利点です。

このうち初心者が使うには、操作性の点からいって「RUN/C」または「Quick C」か「Turbo C」が手ごろでしょう（次項参照）。

・・・コンパイラとインタプリタ

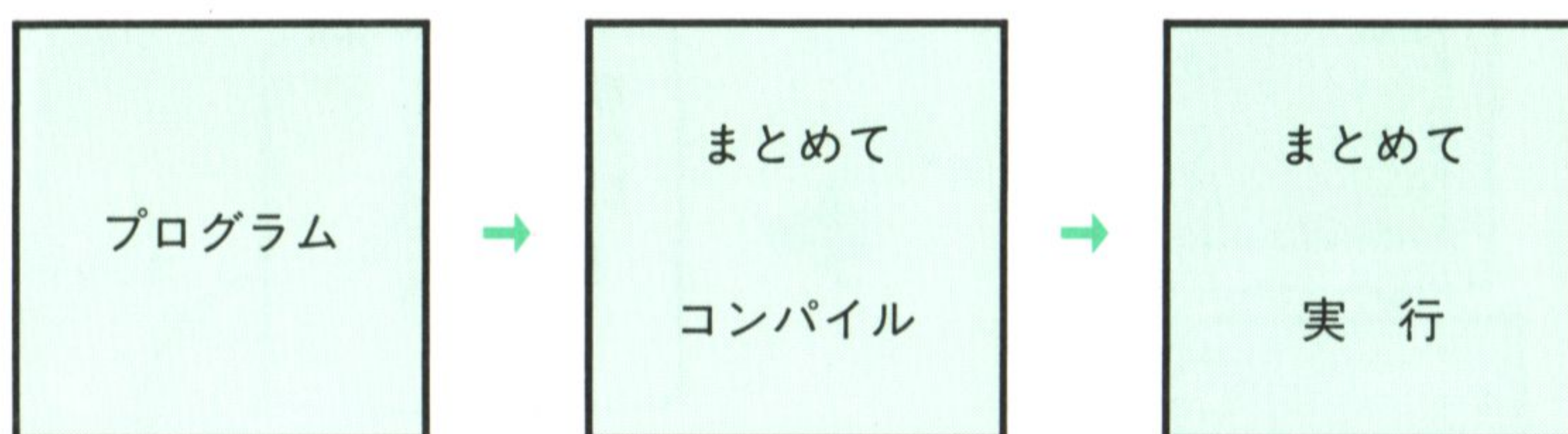
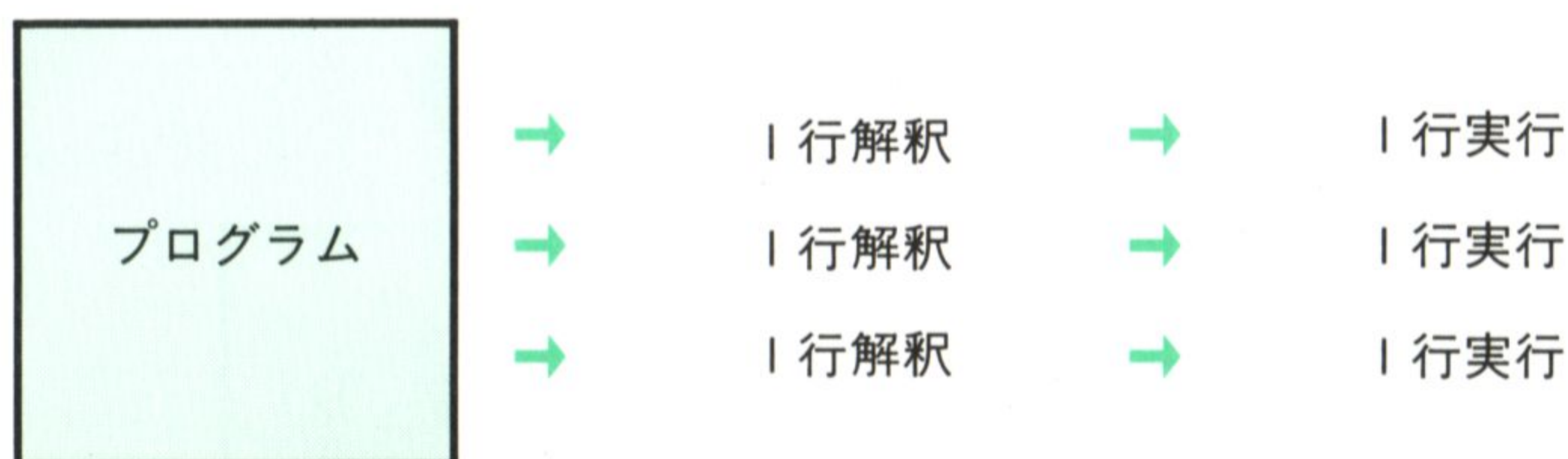
6 実行方式の違い

コンパイラとかインタプリタというのは、C 言語で書かれたプログラムをコンピュータの言葉に翻訳する方式のことです。プログラムはコンパイラによって機械語に変えられ、コンピュータはその機械語の命令によって動きます。この機械語に変える方式の違いで、インタプリタとコンパイラとに分かれているわけです。コンパイラは、プログラムをまとめて機械語に直し、あとでまとめて実行します。インタプリタは、プログラムを1行ずつ解釈しては、その1行を実行するという方式をとっています。

●コンパイラの場合

プログラム	→	まとめて コンパイル	→	まとめて 実　行

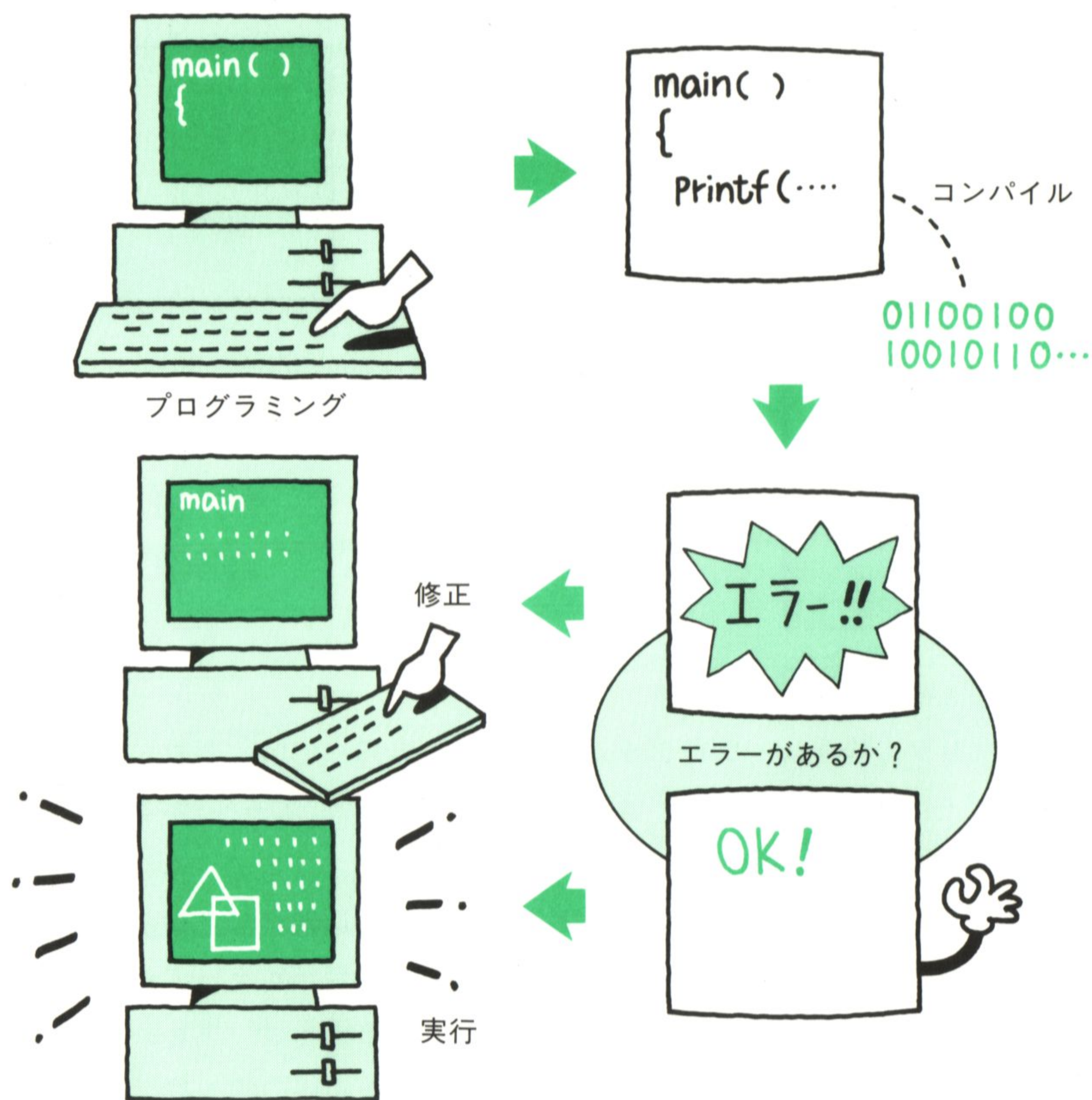
●インタプリタの場合

プログラム				
	→	1行解釈	→	1行実行
	→	1行解釈	→	1行実行
	→	1行解釈	→	1行実行

▲コンパイラとインタプリタ

インタプリタの場合は、プログラムを1行ずつ実行していくので、プログラムの実行スピードは遅くなります。またプログラムに誤り(エラー)があると、その行でプログラムの実行がストップします。このことは、プログラムを実行しながらまちがいを指摘してくれるということで、初心者には便利なところです。ところが、ある程度プログラミングに習熟してくると、これがかえってわずらわしく感じられるようになります。大きなプログラムを作る場合は、まとめてプログラムの誤りを修正しておいてから実行したほうが能率がよいので、専門のプログラマは、コンパイラを使って作業することを好むようです。

つまりプログラムが小さいうちは、実行スピードや手間はインタプリタのほうが優れていますが、何百行、何千行とプログラムが大きくなってくると、インタプリタではまかないきれなくなり、コンパイラのほうが便利で機能的だと感じられるようになるのです。

コンパイラは、プログラムを1行ずつではなく全部まとめて機械語に直します。これを「コンパイルする」といいます。コンパイルの時点でプログラムにエラーがあれば、チェックしてエラーメッセージを出します。エラーがある場合は、機械語には直されません。エラーがなくなれば、そのプログラムは実行することができます。ですから、プログラムを作ってエラーがなくなるまでの段階は手間がかかりますが、その後何度もくり返したいときは、いちいちコンパイルしなくてもよいぶん実行スピードがアップします。

main( )
{
Printf(····
コンパイル
01100100
10010110···
プログラミング
main
修正
エラー!!
エラーがあるか？
OK!
実行

▲コンパイラ形式のプログラムの実行

6 ソフトの操作性の違い

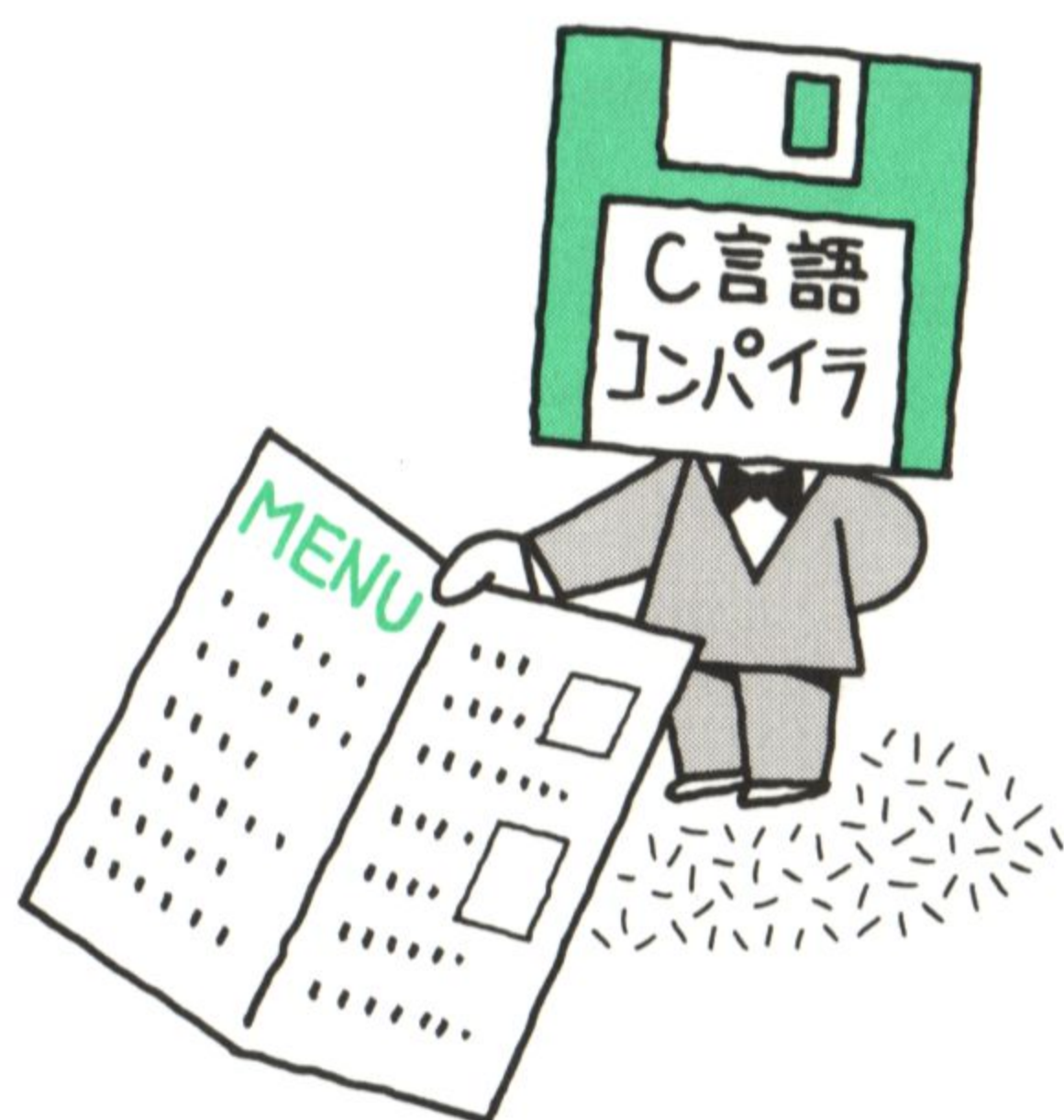
インタプリタの場合、使うのはとても簡単です。たとえば「RUN/C」はパソコン用のインタプリタであるBASICに似せて作られたため、BASICとほとんど同じ操作で使うことができます。

コンパイラの場合は、操作がそれほど簡単ではなく、C言語のコンパイラを購入しても、インタプリタほど手軽にプログラムを動かすことはできません。多くの場合、C言語のコンパイラを動かすには、あらかじめユーザが条件を設定する必要があります。使いたい条件に合わせて準備を行ったあと、ようやくプログラムのコンパイルや、実行をすることができるのです。

また、以前は、C言語のコンパイラにはプログラムを作る機能が付属していませんでした。そこで、別にエディタと呼ばれるソフトを利用してプログラムを作り、コンパイラでそれをコンパイルしていたのです。つまり、エディタ、コンパイラという別の2つのソフトが必要でした。

そのあたりの不便さを解消するために開発されたのが、統合開発環境を持ったコンパイラです。前項で紹介した「Quick C」や「Turbo C」などがそれで、ソフトを起動すると画面に一連のメニューが表示され、「プログラムを作る」、「コンパイルする」、「実行する」などという操作を、ワンタッチで選べるようになりました。あらかじめめんどうな手順を踏む必要がなく、上級者でなくても簡単に使えるようになったのです。

初心者には、インタプリタがお勧めです。入門者レベルのC言語プログラムを作って実行するには、「RUN/C」は手軽で使いやすいソフトです。

たくさんプログラムを作っていこうという人は、インタプリタでは物足りないでしょう。特に実用的なプログラムを作る場合には、インタプリタ方式で毎回1行ずつプログラムを実行していくのは時間がかかってしかたがありません。また、多くのプログラミング言語はコンパイル方式をとっているので、コンパイラを使うことに慣れておく必要もあります。そういう人は、最初からC言語のコンパイラを使っていくほうがよいでしょう。

C言語のプログラミング

プログラムの入力

プログラムの実行手順をもう一度整理すると、①プログラムを入力し、②それを機械語に直し、③実行する、という3つのステップになります。プログラムの筋道を考えてフローチャートを作り、コーディング用紙にプログラムを記述したら、いよいよ①のプログラム入力作業です。これには、エディタと呼ばれるワープロに似たソフト、または「一太郎」など普通のワープロソフトを使用します。

```
B:¥SISOKU.C            [    1:    1]       <挿入>         HELPでヘルプ
/* 四則演算のプログラム */↓
↓
#include <stdio.h>↓
main()↓
{↓
        float x, y;↓
        char a;↓
↓
        printf("四則演算を行います¥n");↓
↓
        printf("演算の種類を1.～4.の数字で入力してください¥n");↓
        printf("1.足し算 2.引き算 3.掛け算 4.割り算  ? ");↓
        a = getchar();↓
↓
        printf(" x = ");↓
        scanf("%f", &x);↓
        printf(" y = ");↓
        scanf("%f", &y);↓
↓
        switch (a) {↓
                case '1' :printf(" x + y = %6.2f¥n", x + y);↓
                          break;↓
                case '2' :printf(" x - y = %6.2f¥n", x - y);↓
  MENU1 MENU2 MENU3 SPLIT FIND   SEL  CUT  COPY  PASTE TAGJP
```

▲エディタによるプログラムの作成（MIFES）

プログラムができあがったら、これにファイル名をつけてMS-DOSのテキスト・ファイル形式でディスクに保存します。このファイルをソース・プログラム、または単にソースと呼びます。ファイル名は、次のように、英字、数字合わせて半角8文字以内でつけ、そのあとに「．」(ピリオド)と拡張子「C」をつけます。

○○○○○○○○．C

ファイル名に用いる英字は、大文字でも小文字でもかまいません。コンパイラ

ピリオド

PROG1-1.C

ファイル名
(半角8文字以内)

拡張子

のほうは、どちらでつけても大文字として判断します（以降、本書で用いるファイル名は、大文字でも小文字でも同じ)。 漢字、ひらがななどの日本語を使うこともできますが、入力がめんどうなのと、システムによっては誤動作の原因ともなるので、ファイル名に日本語を使うのはできるだけ避けましょう。なお、拡張子は、そのファイルの属性（意味や性質）を示すものです。「C」をつけておくと、コンパイラが自動的にC言語のプログラムだと判別してくれます。

ソース・プログラムを機械語に直すには、さらに2つのステップが必要です。その第1はコンパイル、第2はリンクです。コンパイルはソース・プログラムを機械語に翻訳する作業で、コンパイラが行います。しかしこれだけでは、まだプログラムを動かすことはできません。プログラムを実行するために必要な部品をそろえたり、実行時のメモリ上の番地を割りふったりするリンクという作業が必要です。それでようやく、プログラムが実行できる状態になります。

コンパイル

ソース・プログラムを保存したら、「コンピュータの言葉＝機械語」に翻訳するためにコンパイラを作動させます。「Lattice C」、「MS-C」などのコンパイラは、プログラムを機械語に直す働きだけをし、プログラムを書いて保存するエディタの働きは含んでいません。

ドライブ1にコンパイラの入っているディスク、ドライブ2にデータ・ディスクを入れて操作していきます。ここでは、ソース・プログラムを「sisoku.c」というファイル名で保存したことにします。

コンパイラを起動すると、画面にはメッセージが表示されて、コンパイルが行われます。プログラムに文法的な誤りがあれば、コンパイラはそれをチェックし、エラーメッセージを出して知らせます。エラーメッセージは、たとえば次ページの例のようなものです。

```
C>lc -ms -e b:sisoku  > err1
Lattice C ﾊﾟｰｿﾅﾙ, Version 1.0
Copyright 1987      Lattice, Inc.  All rights reserved.
Compiler return code 1

C>if errorlevel 1 goto end

C>
```

▲**コンパイル時のメッセージ（Lattice C）**

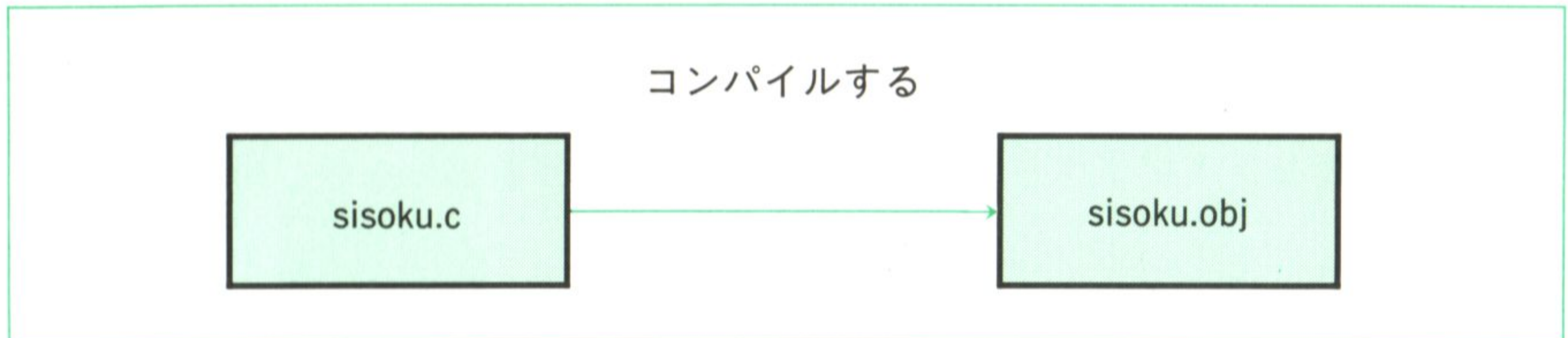
```
C>type err1
b:SISOKU.c 15 Error 57: semi-colon expected
b:SISOKU.c 30 Warning 85: function return value mismatch

Compiling b:SISOKU.C
Compiler return code 1

Total files: 1, Compiled OK: 0, Failed to compile: 1
```

▲**エラーメッセージの例（Lattice C）**

エラーメッセージには、エラーのあるファイル名、行番号などが表示されますから、どこに誤りがあったか、だいたいの見当をつけることができます。例の場合、エディタで「sisoku.c」を呼び出して行15のあたりをみると、たしかにまちがいのあることがわかります。

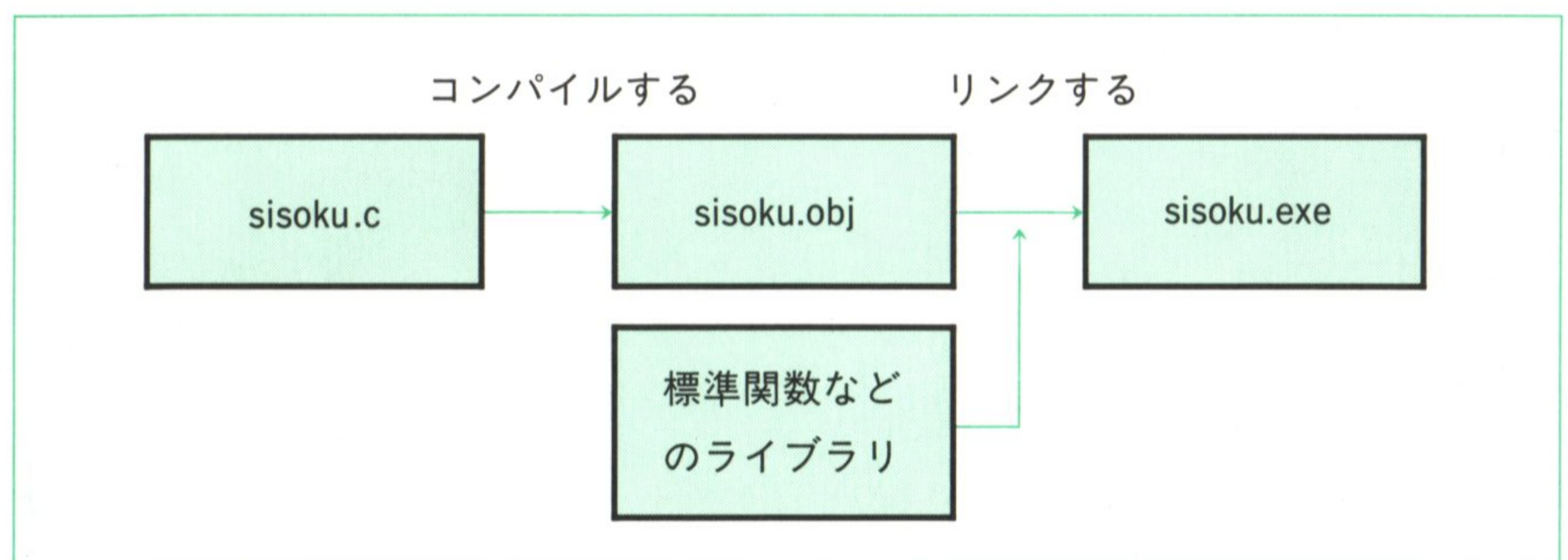
```
B:¥SISOKU.C                    [   15:    1]     <挿入>            HELPでヘルプ
main()↓
{↓
        float x, y;↓
        char a;↓
↓
        printf("四則演算を行います¥n");↓
↓
        printf("演算の種類を1.～4.の数字で入力してください¥n");↓
        printf("1.足し算 2.引き算 3.掛け算 4.割り算  ? ");↓
        a = getchar()↓  ←─┐
↓                          │
■       printf(" x = ");↓  └──この行にセミコロンがない
        scanf("%f", &x);↓
        printf(" y = ");↓
        scanf("%f", &y);↓
↓
        switch (a) {↓
                case '1' :printf(" x + y = %6.2f¥n", x + y);↓
                          break;↓
                case '2' :printf(" x - y = %6.2f¥n", x - y);↓
                          break;↓
                case '3' :printf(" x * y = %6.2f¥n", x * y);↓
                          break;↓
  MENU1 MENU2 MENU3 SPLIT FIND    SEL  CUT  COPY  PASTE TAGJP
```

▲**プログラムのエラー例**

エラーを修正してもう一度「sisoku.c」を上書き保存し、再度コンパイラを動かします。プログラムに誤りがなくなれば、ここで機械語のファイルが作られます。このとき、できたファイルには、自動的に「sisoku.obj」という名前がつけられます。

拡張子「obj」のファイルは、まだ完全な機械語のファイルではありません。これは「リロケータブル・オブジェクト・ファイル」と呼ばれるものです。舌をかみそうな名前で、普通は省略してオブジェクト・ファイルまたはオブジェクトと呼ばれています。

オブジェクト・ファイルは、まだ実行できる段階ではありません。さらにいくつかの必要な部品を取り込み、そのうえコンピュータに正しく命令を与えるための正しい位置にセットして、ようやく動かすことができます。それを行うのが、次のステップであるリンクです。

リンク

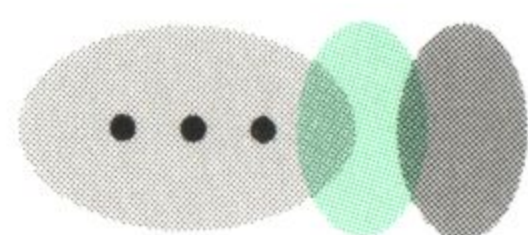
リンク作業を行うには、リンカと呼ばれるプログラム・ファイルが必要です。リンカはコンパイラには含まれていない場合が多いので、通常はMS-DOSのシステムに入っている「LINK. EXE」というファイルをコピーして使います。このファイルをコンパイラのディスクにコピーして使うか、リンク用のディスクを専用に作るかのどちらかになります。

リンクという言葉には、つなぎ合わせるという意味があります。つまり、実行できるプログラムを作るために、必要な部品を用意してつなぎ合わせる作業がリンクです。たとえば「sisoku.c」には

printf

という命令が多く用いられています。これは「” ”」ではさまれた文字を画面に表示させる働きをしますが、この命令は標準関数のライブラリの中に入っています。そこで、必要なライブラリの中からprintfの働きをする部分を呼び出さなければなりません。

また作成したプログラムが大きい場合は、1つのファイルだけではなく、2つ、3つのファイルに分けておく場合も多く、それらはリンクの段階で組み合わされます。

こうして必要なものがそろったら、「obj」ファイルにコンピュータのメインメモリの実際の番地を割り当て、実行ファイルが作られます。実行ファイルには拡張子「EXE」がつけられ、たとえば次のファイル名になります。

sisoku.exe

・・・プログラムの実行

実行ファイル（EXEファイル）ができたら、実行のための準備は、これですべて終了したことになります。プログラム「sisoku.exe」の実行は、MS-DOS(エムエス・ドス)のプロンプト表示の状態から、次のように入力するだけです（ドライブBにsisoku.exeがあるときは先頭にドライブ番号b:をつけます）。

☞ **A>b:sisoku** ⏎ ←リターンキーを押す

```
C>link s¥c b:sisoku ,b:sisoku,,lcm lc

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985


C>
C>b:sisoku
四則演算を行います
演算の種類を1.～4.の数字で入力してください
1.足し算 2.引き算 3.掛け算 4.割り算  ? 3
 x = 28
 y = 7
 x * y = 196.00
```

▲「sisoku」の実行結果

コラム

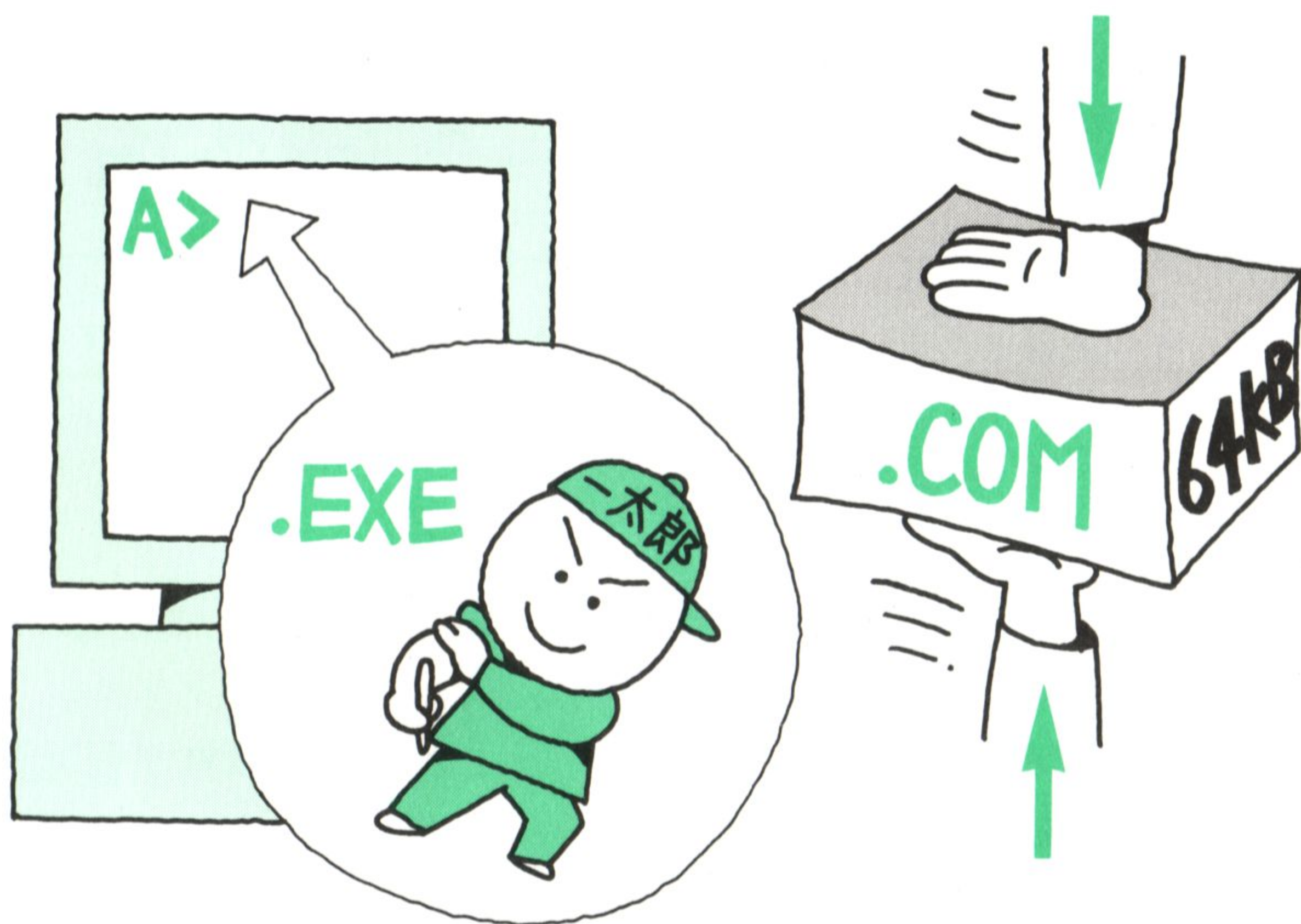
EXEファイルとCOMファイル

「一太郎」などのパッケージソフトのプログラムも、EXE ファイルのひとつ。EXE ファイルのことを、「実行可能形式のファイル」または「実行ファイル」という呼びかたをする。拡張子が「EXE」とつけられているファイルは、MS-DOS のコマンドレベル（プロンプトが表示されている状態）から、ファイル名を入力するだけで実行することができる（拡張子は省略してよい）。

一太郎の起動で「A>JXW」としたのは、つまりこのファイルを呼び出して実行しているわけだ。

実行可能形式のファイルには、EXE ファイルのほかにもうひとつあり、拡張子「COM」のつけられた COM ファイルがそれにあたる。これは、EXE ファイルを 64KB（キロバイト）以内の大きさに圧縮したもの。COM ファイルの代表例としては、MS-DOS で入力されたコマンドを解釈、実行する COMMAND.COM がある。

通常、C 言語のコンパイラでは EXE ファイルが作られる。COM ファイルは、プログラムの大きさによって、作ることができる場合とできない場合がある。

それが可能な場合に COM ファイルを作るには、MS-DOS のコマンド・ファイル「EXE2BIN.EXE」を用いる。

タイプ別C言語ソフトの操作性

C言語でプログラムを作るための主なソフトについては、38ページで簡単に紹介しました。「プログラムを書く→コンパイルする→リンクする→実行する」というプログラミングの4ステップのうち、「MS-C」、「Lattice C」といったコンパイラは、「コンパイルする」ためだけのソフトです。ですから、コンパイラのほかに、プログラムを書くためのエディタまたはワープロソフト、リンクするためのリンカが必要だったわけです(もちろん、それらが動く土台となるOSであるMS(エムエス)-DOS(・ドス)のシステムディスクも必要)。

ところが、これでは実際に使えるための準備を整えるまでの手順がとても煩雑(はんざつ)で、初心者はC言語の勉強を始める前の段階で、すでにギブアップするという状態でした。そこで、とりあえず初心者用に一式取りそろえて提供しよう、というシステムが開発されました。コンパイラとエディタなどをまとめて操作を簡単にしたもので（これを、統合開発環境という)、「Quick C」や「Turbo C」、インタプリタの「RUN/C」などがあります。

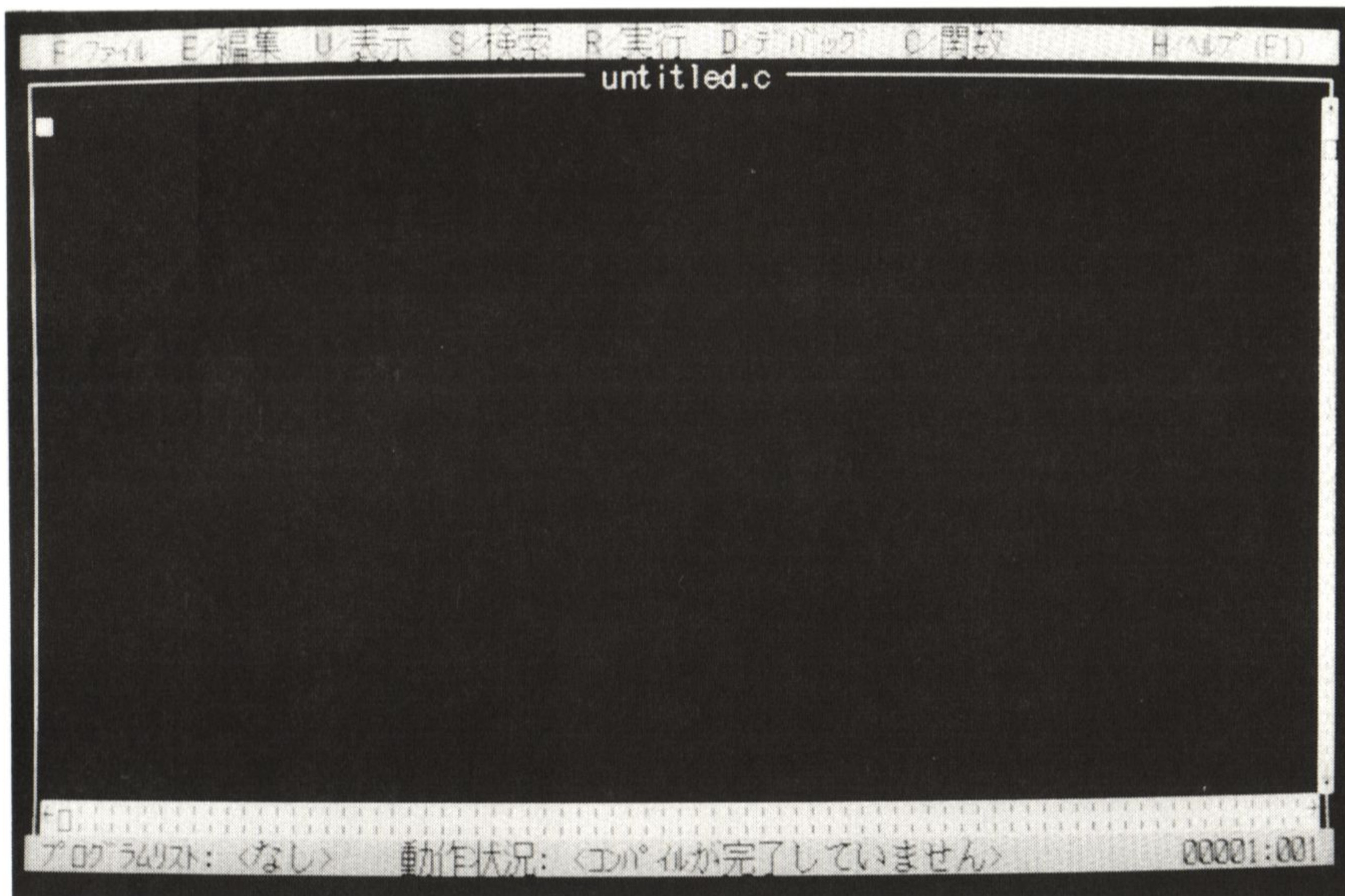
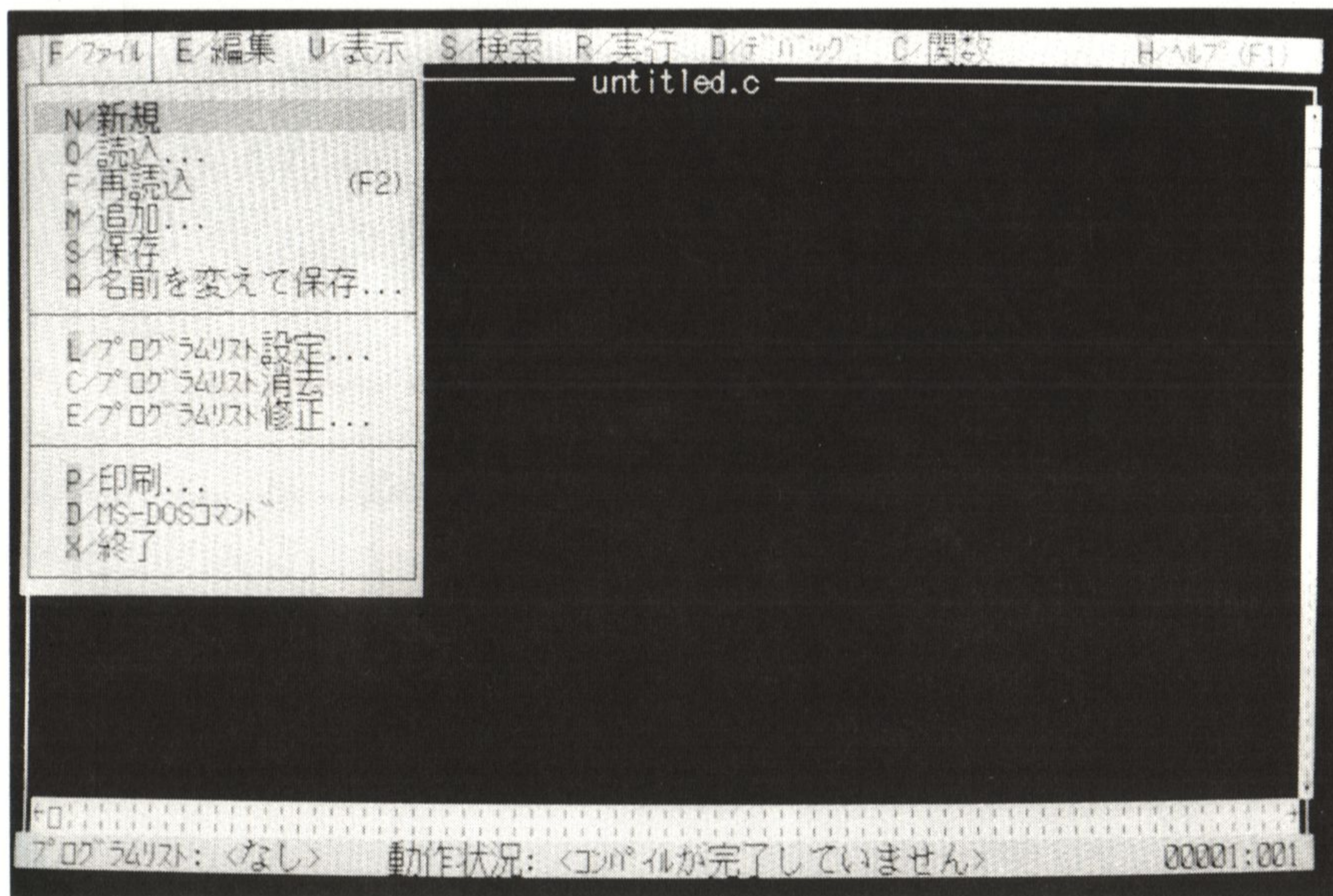
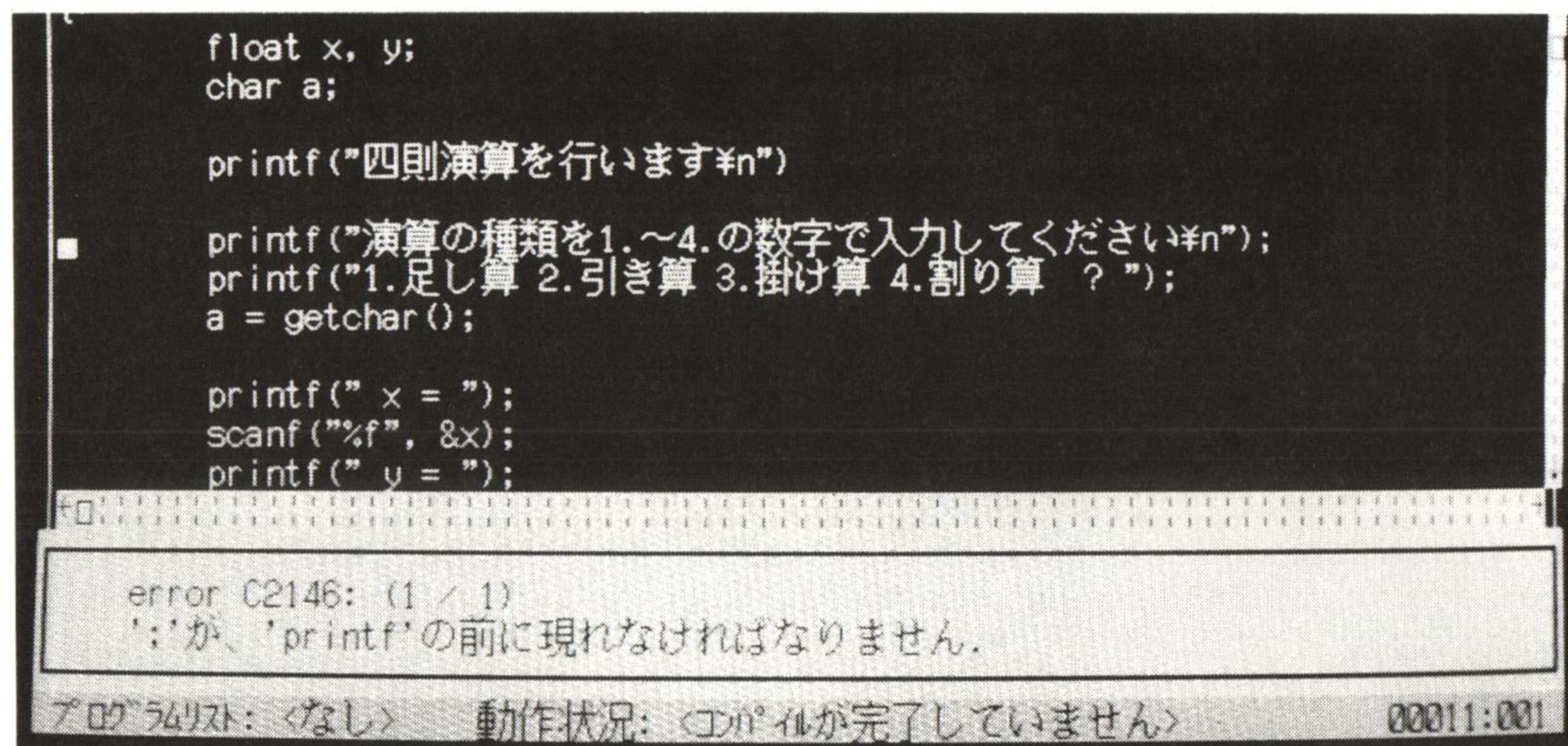
以下、統合開発環境ソフト、インタプリタ、コンパイラ専用ソフトの順に、それぞれの操作性を簡単にまとめておきます。

統合開発環境ソフト――Quick C、Turbo C

統合開発環境ソフトでは、プログラムの入力、コンパイル、リンク、実行という手順が、画面を見ながらだれでも簡単に操作できるようなくふうがなされています。「Quick C」を例に話を進めますが、「Turbo C」でも操作方法はほとんど同じです。

6 機能の選択

「Quick C」を起動すると、次ページ上のような画面が現れます。この画面が、そのままエディタになります。画面の最上行に表示されている「F/ファイル、E/編集、R/実行」などが、「Quick C」の主なメニューです。この状態で GRPH キーを押しながら、先頭に表示されている英字のキーを押すと、それぞれの機能のサブメニューが、画面上に引き下げられた形でオープンされます（このようなメニューをプルダウン・メニューという)。

下の写真は、GRPH キーを押しながら F キーを押して、「ファイル」メニューを開いたところです。

▲ **Quick C の起動画面**

▲ **「F/ファイル」のプルダウン・メニュー**

メニューの選択は、キーボードから選びたい機能の頭文字を入力するか、マウスを使用します。マウスを使えば、マウスカーソルを各コマンドに合わせるだけで、それぞれの機能を選択することができます。

6 プログラムの作成と実行

プログラムを作るには、「E/編集」を選んで画面上にプログラムを書き、続けて「R/実行」で実行します。統合開発環境の特徴は、プログラムの保存もコンパイルもリンクも考えずに、ただ「実行」を選ぶだけですむという点にあります。

画面上のプログラムは、ファイルに保存しなくても実行することができますが、のちに使用するのなら保存しておいたほうがよいでしょう。そのときは、「F/ファイル」のプルダウン・メニューから「S/保存」でプログラムを保存することができます。

「R/実行」を選ぶと、「Quick C」の内部では、コンパイル→リンクという手順が自動的に行われたのち、プログラムが実行されます。プログラムにエラーがあるときは、画面の最下行の欄にエラーメッセージが表示されるので、それを見ながらプログラムを修正します。

```
        float x, y;
        char a;

        printf("四則演算を行います¥n")

        printf("演算の種類を1.～4.の数字で入力してください¥n");
        printf("1.足し算 2.引き算 3.掛け算 4.割り算  ? ");
        a = getchar();

        printf(" x = ");
        scanf("%f", &x);
        printf(" y = ");

    error C2146: (1 / 1)
    ';'が、'printf'の前に現れなければなりません.

プログラムリスト: <なし>    動作状況: <コンパイルが完了していません>    00011:001
```

▲ Quick C のエラー画面

うまく実行できるときは、コンパイル中であることを示すメッセージが表示されたのち、画面が変わって実行結果が表示されます。このあと、↵ キーを押せば、もとの「Quick C」の画面に戻ります。

途中で操作がわからないときは、f・1 キーを押すと、ヘルプメニューが表示されます。必要な項目を選ぶと操作方法が説明されますから、あわててマニュアルを探さなくてもおおよその手順がわかるようになっています。

```
F/ファイル E/編集 U/表示 S/検索 R/実行 D/デバッグ C/関数            H/ヘルプ(F1)

プログラムリスト:    <なし>
カレントファイル :    B:¥sisoku.c

  警告レベル           出力オプション               その他の設定
   ( ) 0/レベル0        ( ) O/OBJ                 [ ] D/デバッグ
   (•) 1/レベル1        (•) M/メモリ              [ ] P/ポインタチェック
   ( ) 2/レベル2        ( ) E/EXE                 [X] S/スタックチェック
   ( ) 3/レベル3        ( ) Y/構文チェックのみ     [X] L/拡張仕様
                                                   [ ] Z/最適化

I/インクルード:
                                  Line:  128
F/デファイン:                     Error: 0

B/更新ファイルのコンパイル  C/カレントファイルのコンパイル  R/全ファイルのコンパイル  取消

プログラムリスト: <なし>   動作状況: <コンパイル完了:実行できます>          00009:001
```

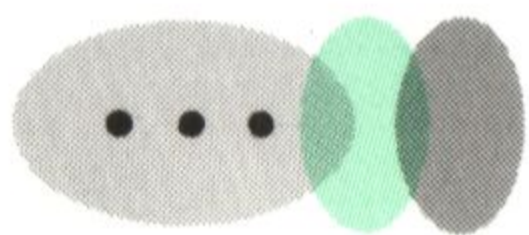
▲コンパイル中のメッセージ

```
プログラムが(16)を返しました.  何かキーを押してください
四則演算を行います
演算の種類を1.～4.の数字で入力してください
1.足し算 2.引き算 3.掛け算 4.割り算  ? 3
 x = 5
 y = 3
 x * y =  15.00

プログラムが(16)を返しました.  何かキーを押してください

  C1   CU   CA   S1   SU     VOID  NWL  INS  REP  ^Z
```

▲実行結果

6 統合開発環境とコマンドライン

統合開発環境は、初心者にはとても便利なものです。ところが、たくさんのプログラムを組むプログラマには、こういった環境は必ずしも評判がよいとはかぎりません。もっと優れたエディタを使いたいとか、コンパイルしただけで保存したいとか、いろいろな細かい要請に統合開発環境が対応しきれないからです。

そのため「Quick C」や「Turbo C」では、別に、コマンドで自由自在に条件を指定するコマンドラインと呼ばれる方法も選択できるようになっています。初心者には統合開発環境、熟練のプログラマのためにはコマンドラインという2つの方法を提供しているのが、これらのコンパイラの大きな長所といえるでしょう。

インタプリタ――RUN/C

6 プログラムの作成と実行

インタプリタの代表は、N-88BASIC(ベーシック)です。PC-9800 シリーズのパソコンの電源を入れるだけで出てくるのがこの BASIC で、そのままプログラムを入力→実行することができます。

プログラムの実行は、「RUN(ラン)」と命令を与えるだけです。プログラムにエラーがある場合は、エラーの前まで実行したあと、エラーメッセージを表示してストップします。書かれたプログラムを1行ずつ実行するというインタプリタ形式は、BASIC ならではの特徴です（BASIC のコンパイラもある）。

◉ BASIC のプログラム例――文字を表示する

```
OK
10 PRINT "a"
20 PRINT "b"
30 PRINT "c"
OK
RUN ↵
a
b
c
OK
```

- 10〜30行 ← 行番号がついている部分がプログラム
- RUN ↵ ← 「実行せよ」という命令
- a, b, c ← 実行結果

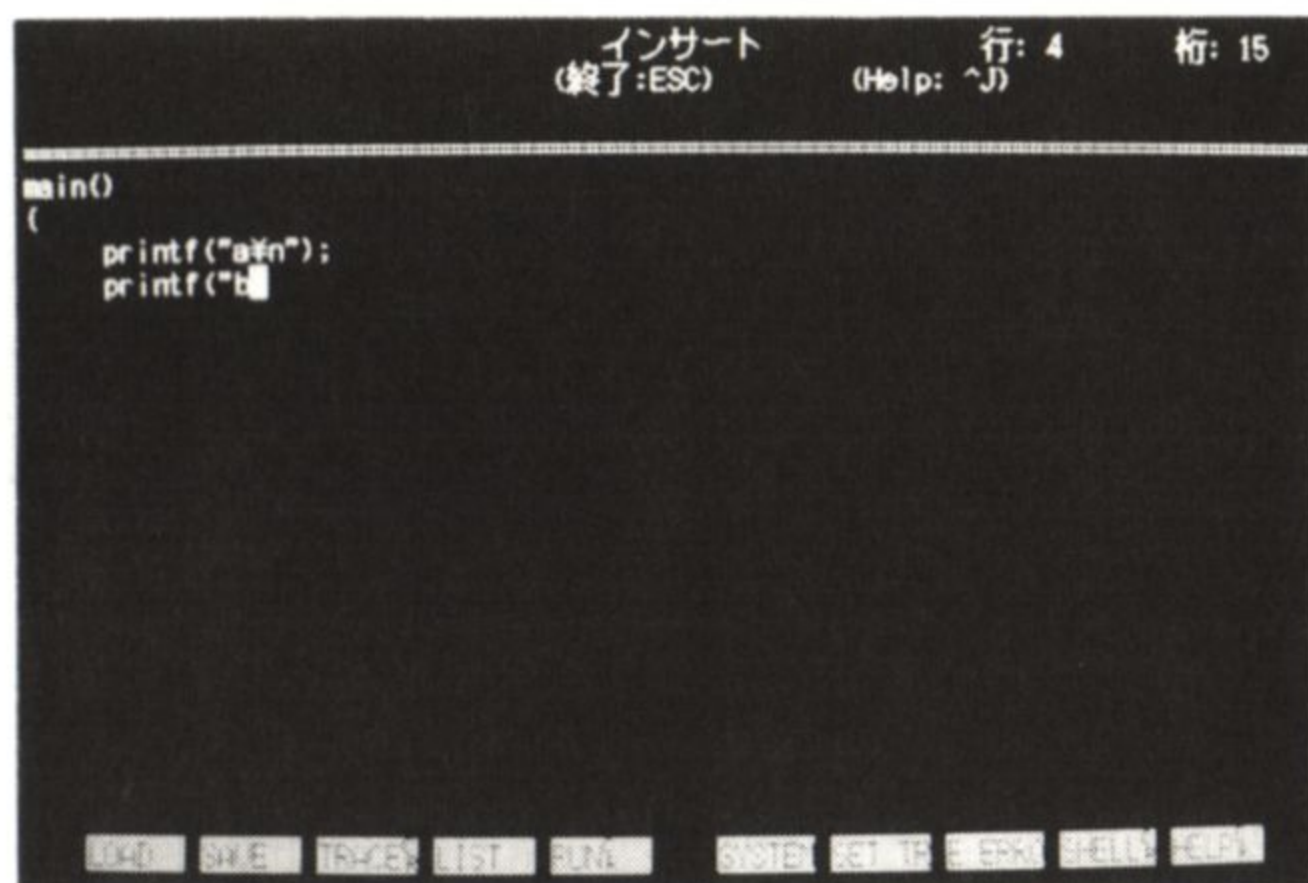
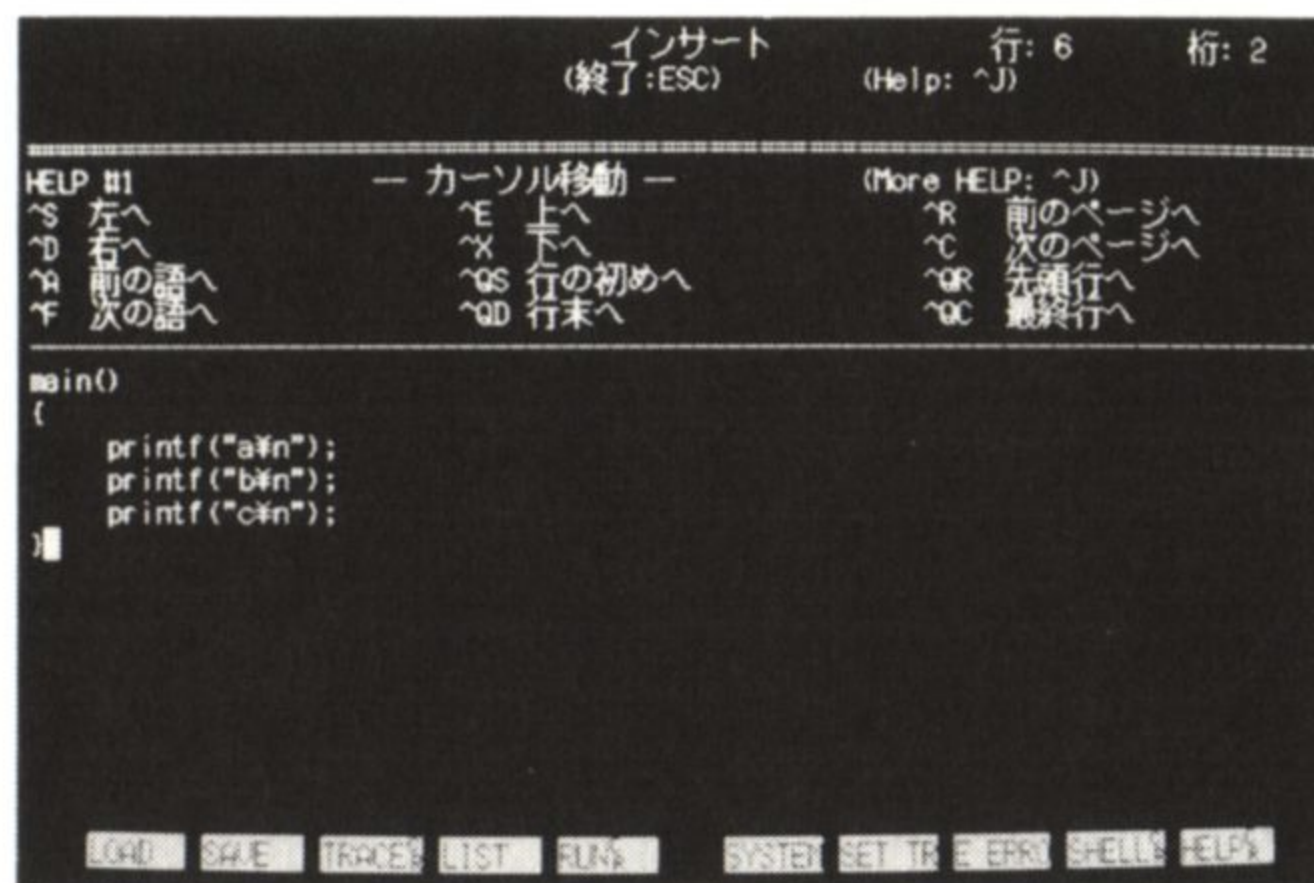
◉エラーのあるプログラム例

```
OK
10 PRINT "a"
20 PPINT "b"
30 PRINT "c"
OK
RUN ↵
a
Syntax error in 20
OK
```

- 20 PPINT "b" ← R が P になっている
- a ← 行10を実行
- Syntax error in 20 ← エラーメッセージ

C言語のソフトにもインタプリタがあり、ライフボート社の「RUN/C」がそれです。BASICとほとんど同じ操作で、C言語のプログラムを走らせることができます。

プログラムの作成は、「EDIT」という命令でエディタ画面を呼び出し、その画面上にプログラムを入力していきます。入力が終われば、ESCキーを押すとエディタ画面から抜け出すので、「RUN」という命令を与えると、プログラムが実行されます。プログラムに誤りがあればその直前まで実行し、エラーメッセージを出してストップします。

BASICで例にあげた「文字を表示する」プログラムを、「RUN/C」でも同様に作成し、実行してみましょう。

▲RUN/Cによるプログラムの入力

▲ヘルプ画面

◎RUN/Cのプログラム例——文字を表示する

```
main()
{
    printf("a¥n");
    printf("b¥n");
    printf("c¥n");
}

OK
run ↵        ←実行命令
a
b
c
OK
```

●エラーのあるプログラム例

```
main()
{
    printf("a¥n");
    print("b¥n");        ←「printf」のfが抜けている
    printf("c¥n");
}

Ok
run [↵]                  ←実行命令
a
… ｴﾗｰ #21 が[4]行で発生しました              ┐
変数または関数'print'がありません            ┘←エラーメッセージ
Ok
```

6 インタプリタの得手、不得手

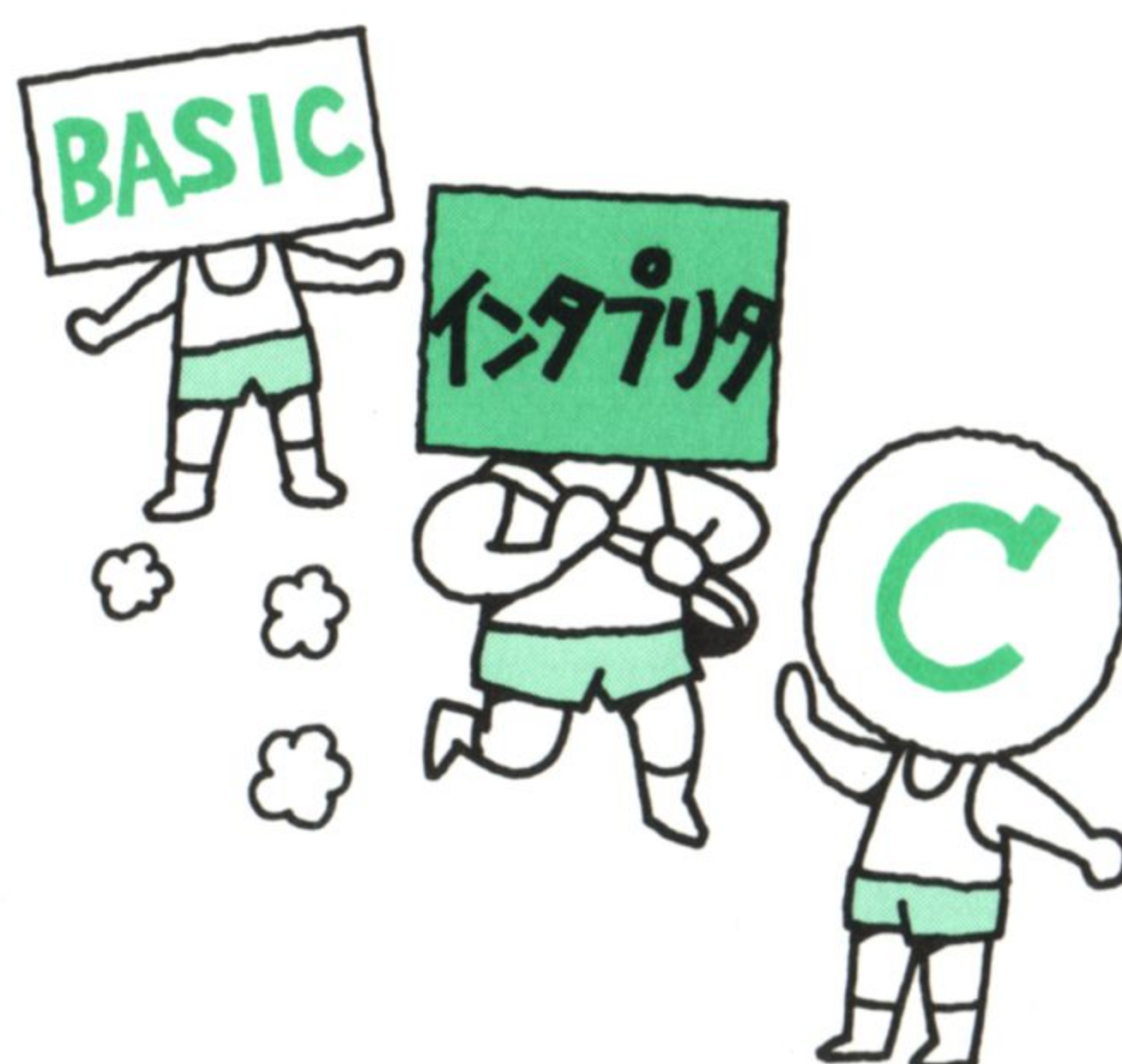
このように、「RUN/C」の操作は、BASICとほとんど同じです。それもそのはず、「RUN/C」はN-88BASICのもとになったマイクロソフト社の「Microsoft BASIC」の仕様に合わせて作られ、BASICに慣れている人がC言語をスムーズに学べるよう、学習用として開発されたものだからです。

インタプリタは操作が簡単で、いくつかのコマンド（RUN、LIST、SAVE、LOADなど）の使いかたを覚えるだけで、すぐにプログラミングを始めることができます。これは、初心者には非常にありがたいことです。また、エディタに打ち込んだプログラムをフロッピィディスクなどに保存しておくと、ほかのC言語コンパイラでも、そのプログラムを呼び出して実行することができます。

このように初心者にはとても扱いやすい「RUN/C」ですが、インタプリタであるための欠点もいくつかあります。まず、インタプリタはプログラムを1行ずつ翻訳して実行していくため、プログラムが大きくなるとスピードが遅くなることです。また、エラーがあると、その行でプログラムがストップします。このことは、エラーを発見しやすい反面、1つひとつエラーを修正するのはめんどうな場合もあります。プログラムのエラーだけをまとめてチェックしたい人には、わずらわしく感じられるのです。

前述したように、プログラムが小さいうちは、実行スピードや手間を考えると

インタプリタのほうが速いのですが、プログラムが大きくなってくると、コンパイラのほうが機能的だと感じられるようになります。

ですからインタプリタは、C言語を覚えるための学習用または入門用だと考えればよいでしょう。とりわけ、BASICを知っている人がC言語の勉強をするのには、とても使いやすいソフトです。

コンパイラ――Lattice C パーソナル

統合開発環境を持っていないコンパイラの場合は、そのまますぐに使うことはできませんので、あらかじめいろいろな条件を整えてコンパイルの準備をしなければなりません。

準備としては、もとのコンパイラのディスクから必要なファイルだけを取り出し、好みのエディタやリンカも合わせて、使いやすいディスク構成にしておくのが便利です。この概略を、「Lattice C パーソナル」の例で説明していきましょう。実際にはコンパイラごとに方法は違いますが、基本的な手順は同じです。なお「Lattice C パーソナル」は、「Lattice C Ver.3.1」を個人ユーザ対象に変え、安価で販売しているものです。

6 実行ディスクの作成

まず、必要なファイルをコピーして、コンパイラ用の実行ディスクを作ります。

❶フロッピィディスクの準備（3枚）

実行ディスクは、以下の3種類が必要です。新しいフロッピィディスクを3枚用意して、それぞれラベルをつけておきます。

・C言語のコンパイラ・ディスク

・エディタとフロントプロセッサ用のディスク

・データ用のディスク

❷コンパイラ・ディスクの作成

ドライブAからMS-DOS（エムエス・ドス）を起動し、コンパイラ・ディスクをドライブBに入れ、FORMAT（フォーマット）コマンドで次のようにフォーマットしておきます。

```
                    ┌────── 1字ぶん（半角）空ける
           A>FORMAT B:/S
ドライブ B のディスクをフォーマット ┘  └─ MS-DOS の基本システムをコピー
```

フォーマットがすんだら、次の各ファイルを、「Lattice C パーソナル」からコピーします。

LC.EXE LC1.EXE LC2.EXE	この３つはコンパイラの本体
ディレクトリ INCLUDE の中身全部	ヘッダファイルの集まり
ディレクトリ S の中身全部	標準関数など、いろいろなライブラリの集まり（スモールモデルの場合）
LINK.EXE	MS-DOS のシステムからリンカをコピー

「Lattice C」には、スモールモデル、ミディアムモデル、ラージモデルなど数種類のメモリモデルが用意されていて、作るプログラムやデータの大きさによって選択します。練習問題程度のプログラムを作る場合は、いちばん小さいスモールモデルを選びます。プログラム領域 64KB、データ領域 64KB 以内のプログラムまでは、このモデルで十分です。モデルが違うと、コピーするライブラリや、コンパイル時の設定が違ってきます。

コピーが終わったら、同じディスクに、手順❸でコピーするエディタを使って、以下のように「C.BAT」というファイルを作っておきます。このファイルは、コンパイルからリンクまでを行うように手順を指定したバッチ・ファイルです。

◉バッチ・ファイル c.bat

```
LC -ms -e %1 > b:err*  ←── コンパイルせよという命令
if errorlevel 1 goto end  ←── エラーがあったときの処理
link s¥c %1,%1,,lcm lc  ←── リンクせよという命令
:end
                   *エラーが出ると、エラーファイル err をドライブBに作る
```

このバッチ・ファイルは

☞ **A>C␣「ファイル名」** ⏎ （␣は半角スペース１字ぶん。以下同じ）

と入力すると、「%1」のところに入力したファイル名が入り、指定のファイルをコンパイルします。何か誤りがあればエラーメッセージを表示してストップし、

正しいプログラムであれば、続けてリンクを行います。

❸エディタ用のディスクの作成

エディタには、「MIFES-98」、「RED++」、「Final」などがあります。ワープロソフトをエディタがわりに使うこともできますが、ワープロを起動してプログラムを作り、終了するにはけっこう時間がかかります。プログラムの作成には、エディタのほうが便利です。説明では、「MIFES-98」を例にします。

これもコンパイラ・ディスク同様、フォーマットと同時に、MS-DOS のシステムをコピーしておきます（オプション・スイッチ「/S」をつける)。

☞ **A>FORMAT␣B:/S ↵**

フォーマットが終了したら、エディタをコピーします。MIFES-98の場合は、次の 2 つのファイルをコピーすることになります。

ファイル	内容
MIFES.EXE	MIFES-98の本体
MIFES.HLP	ヘルプメッセージの入ったファイル

次に、同じディスクに、フロントプロセッサをコピーします。ATOK6 の例では、次の 3 つのファイルになります。

ファイル	内容
ATOK6A.SYS	フロントプロセッサのシステム
ATOK6B.SYS	フロントプロセッサのシステム
ATOK6.DIC	辞書

これだけコピーしたら、次にエディタを起動して、同じディスクに「CONFIG.SYS」「AUTOEXEC.BAT」の 2 つのファイルを作ります。

◎システムファイル CONFIG.SYS とバッチ・ファイル AUTOEXEC.BAT

● CONFIG.SYS

```
FILES=12
BUFFERS=5
DEVICE=ATOK6A.SYS /T=1 /E=1  ←フロントプロセッサを指定
DEVICE=ATOK6B.SYS            ←        〃
```

● AUTOEXEC.BAT

```
set INCLUDE=A:¥INCLUDE  ←ヘッダ・ファイルのあるディレクトリを指定
set LC=A:¥              ←コンパイラのあるディレクトリを指定
set LIB=A:¥s            ←ライブラリ・ファイルのあるディレクトリを指定
```

❹データ用ディスクの作成

最後に、作ったプログラムを保存するための、データ・ディスクを作成します。これは、フォーマットさえしておけばよく、オプション・スイッチ「/S」は必要ありません。

☞ **A>FORMAT␣B: ↵**

❻プログラミングの手順

作成した3枚のディスクを使ってプログラミングする手順は、次のとおりです。

①ドライブAにエディタ・ディスクを入れ、パソコンを起動する。これで、自動的にフロントプロセッサが立ちあがる。

②ドライブBにデータ・ディスクを入れる。

③エディタを起動して、プログラム（例えばPROG1-1.C）を作る。まず次のように入力し、エディタ画面にプログラムを書き込んでいく。

☞ **A>MIFES␣B:PROG1-1.C ↵**

④できあがったプログラムを、ドライブBのデータ・ディスクに保存する。

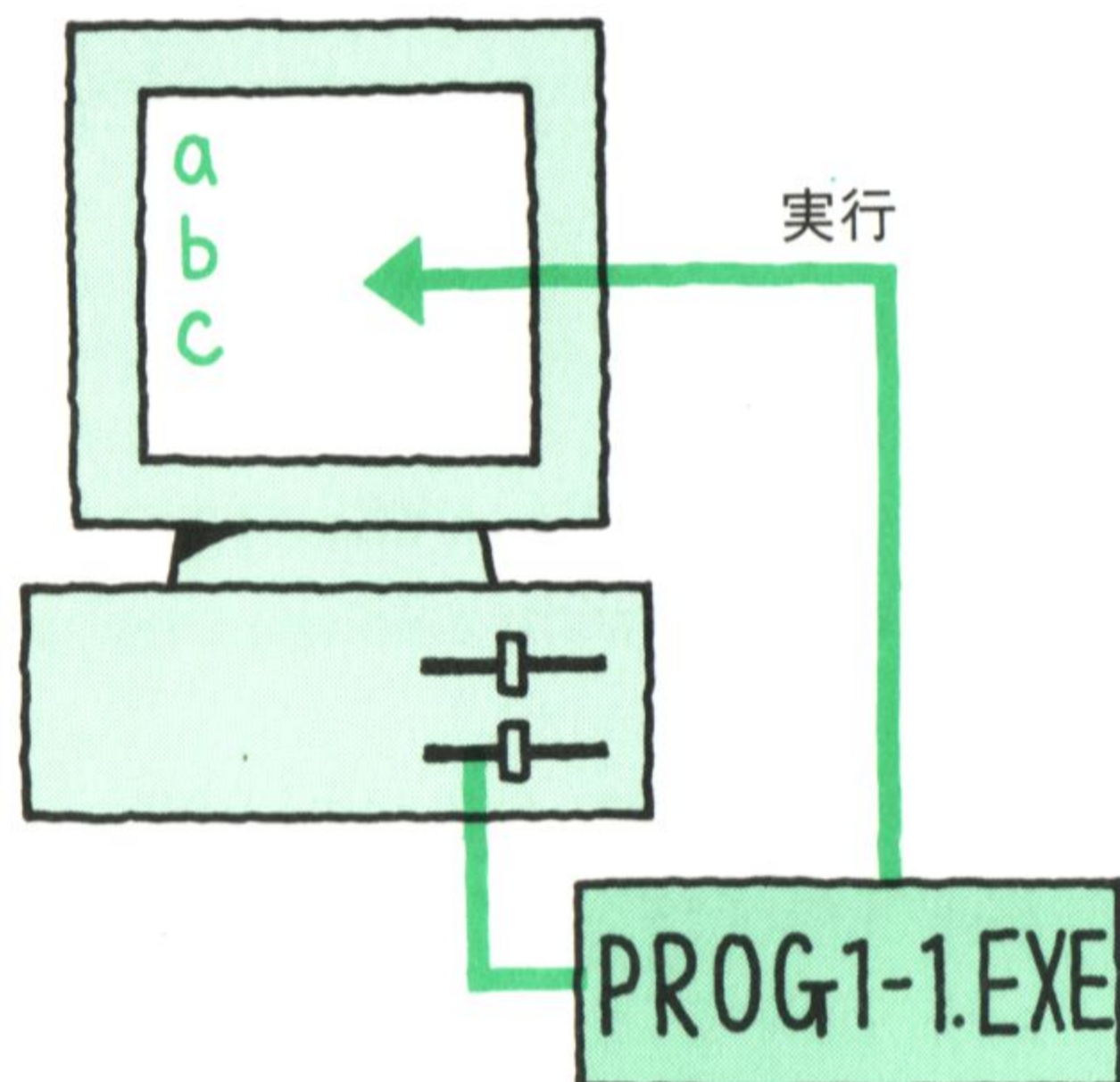
⑤ドライブAのディスクをコンパイラ・ディスクに入れかえ、次のように入力する。これで「C.BAT」ファイルが動き、プログラムをコンパイル、リンクする。

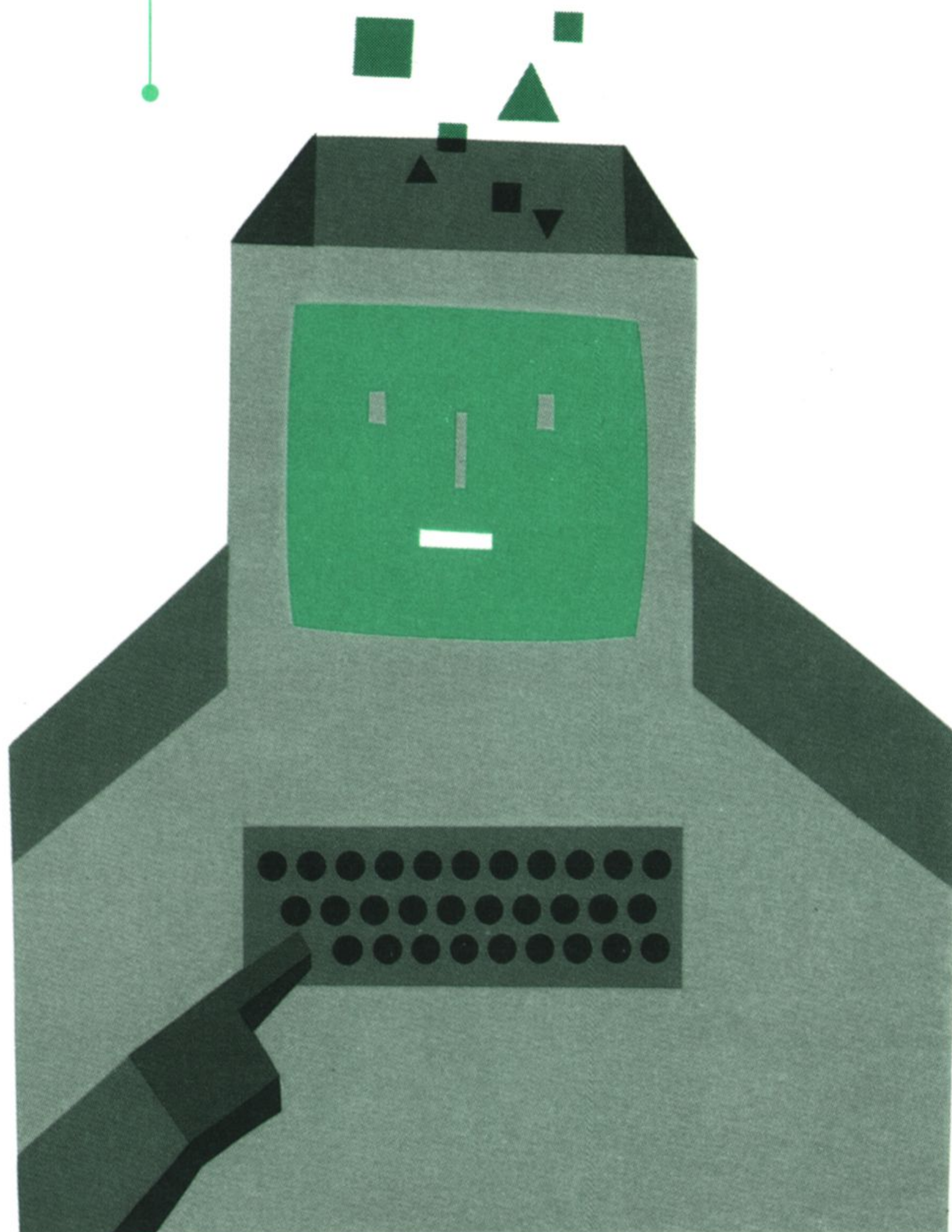
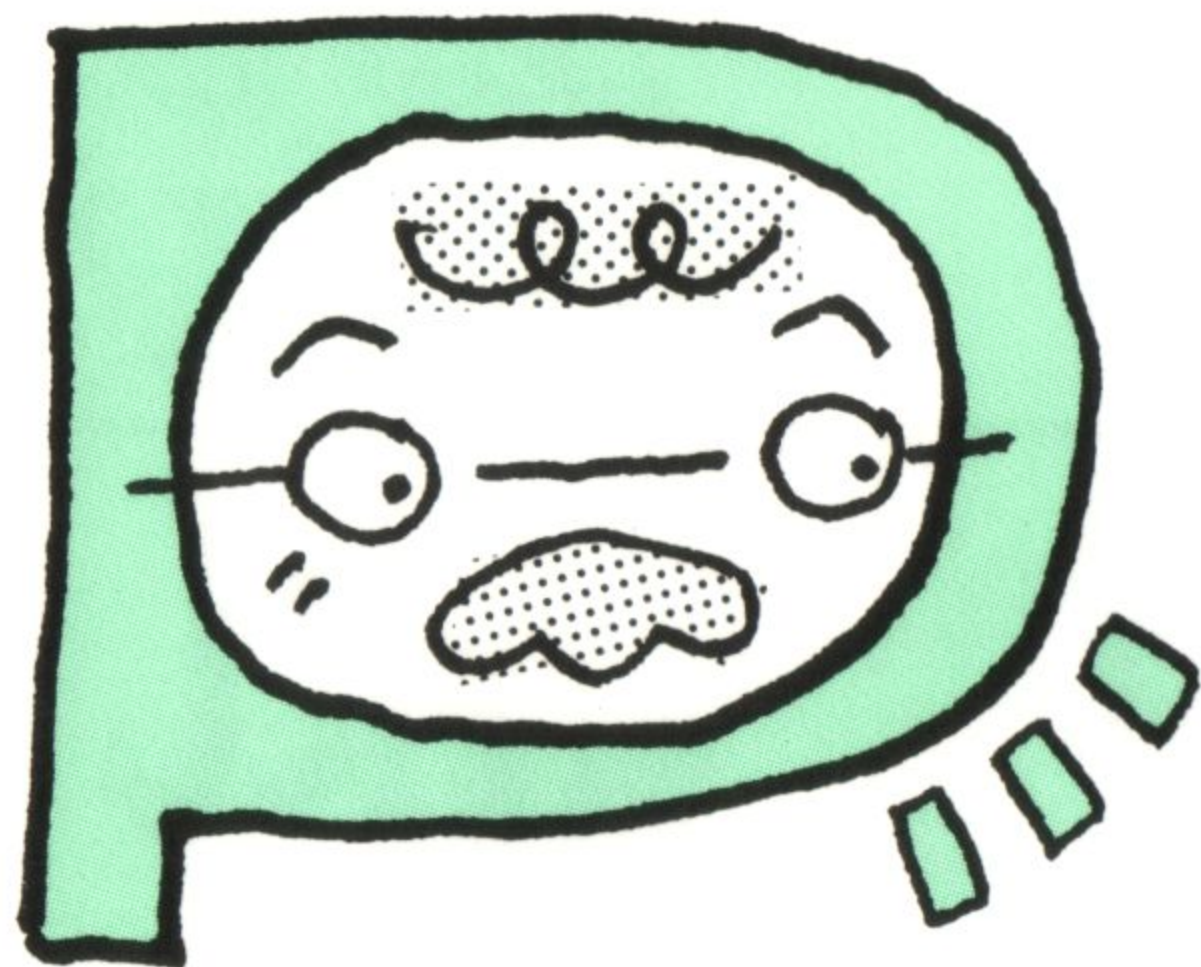
☞ **A>C␣B:PROG1-1 ↵**

⑥エラーが出ずに正しく終了すると、ドライブBに「PROG1-1.OBJ」、「PROG1-1.EXE」の2つのファイルができる。

⑦プログラムを実行する。ファイルのあるドライブ名、ファイル名を指定する。

☞ **A>B:PROG1-1 ↵**

コンパイラやエディタが例にあげたもの以外だと、コピーするファイル名や、「C.BAT」、「CONFIG.SYS」、「AUTOEXEC.BAT」の内容は違ってきますが、手順としてはほぼ同じです。基本的な流れは変わりありません。

PART 3

いよいよプログラムだ……

田川部長、プログラミングに挑戦する。

玲子　どう？　C言語の姿がつかめてきたかしら。

田川　ウーン、わかったような気もするけど、実感が伴わないね。プログラムを作るまでに、知っておかなければならないことがたくさんあるからたいへんだ。

玲子　実際にプログラムをいくつも作って試していくと、感じがつかめてくるものよ。PART2 ではコンパイラとインタプリタの違い、プログラムを書いて実行するまでの手順などを説明したけど、ところで何かソフトを手に入れたかしら？

田川　ほら、「Quick C」を持ってきたぞ。初心者にも簡単に使えそうだし、なにしろ安い。家でちょっと使いかたを勉強してきたよ。

玲子　それはご立派！「Quick C」は画面の表示が親切だから使いやすいでしょう。では、プログラムの話に入りましょう。実際にコンパイラを使って、プログラミングすることにします(読者の皆さんは、C言語のコンパイラかインタプリタを用意して、実際に操作してみてください)。

田川　いよいよ、プログラミングか。武者ぶるいがするぞ。あ、キーボードに触ると手がふるえる……ブルブル、緊張するなあ。

玲子　「Quick C」は、プログラムを入れたらすぐ実行でき、エラーが出てもわかりやすいのでとても簡単なのよ。最初にプログラムをひとつお見せしましょう。次のページのサンプルプログラムがそれよ。

田川　オッ、最初から、むずかしそうなのが出たぞ。お手やわらかに。プログラムというのは、英語と数学みたいなものだねえ。チンプンカンプンだ……。

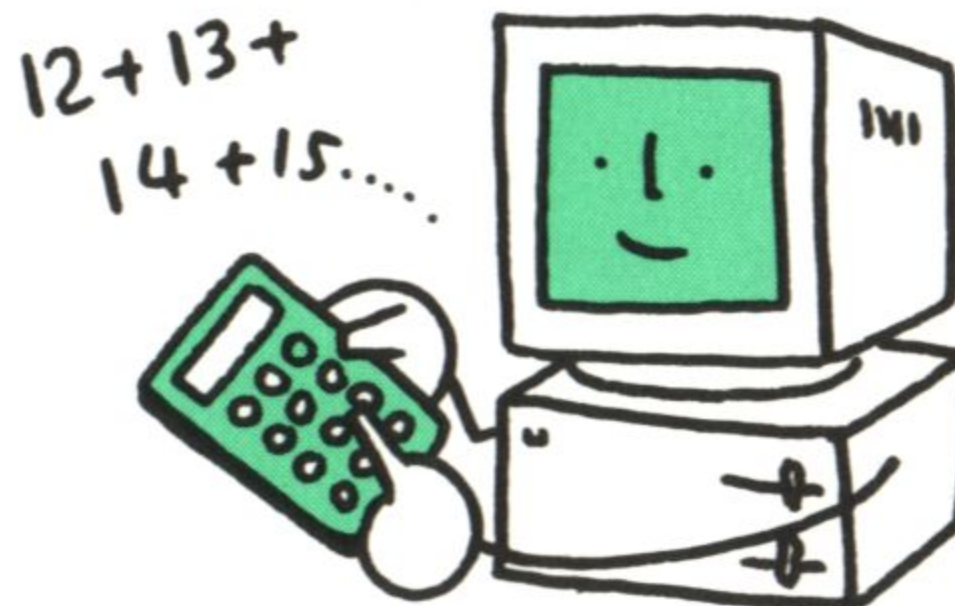
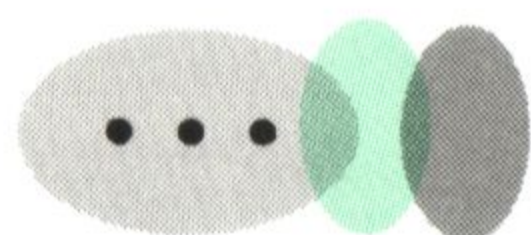
玲子　このプログラムは「和を計算するプログラム」なの。C言語を知らない人にはプログラムは暗号みたいなものだけど、暗号がわかれば意味は自然にわかってくるものよ。それに、内容がわからなくても、実際に動かしてみれば働きだけは理解できるわね。では、このプログラムの暗号を解いていくことにしましょう。

プログラムの入力→実行の実際

本項では、次のサンプルプログラムを例に、プログラムの実行までの手順を説明していきます。

◉サンプルプログラム——和を計算する

```
#include <stdio.h>

main()
{
    int a, b, c, i;

    printf("整数を２つ入力してください¥n例 1,100¥n");

    scanf("%d,%d", &a, &b);

    printf("%dから%dまでの和を計算します¥n", a, b);

    c = 0;
    if ( a <= b )
        for ( i = a; i <= b; i++ )
            c = c + i;

    else
        for ( i = b; i <= a; i++ )
            c = c + i;

    printf("答え  : %d¥n", c);
}
```

12+13+
14+15.....

・・・プログラム入力のコツ

エディタの用意

パソコンとC言語のコンパイラを用意して、まず、サンプルプログラムをキーボードから実際に打ち込んでみましょう。「Quick C」などコンパイラにエディタがついているものは、それを利用してプログラムを打ち込みます。エディタがついていない場合は、「MIFES」など市販のエディタを使います。それもないときは、MS-DOSに付属しているEDLINというエディタを使います。それぞれについては、各マニュアルを参照してください。

文字の入力は、プログラムを見ながら慎重に行います。「{　}」や「(　)」が正しく対になっているか、行の終わりにつけられている「;」(セミコロン)を忘れていないか、文字の綴りをまちがえていないかなどに気をつけましょう。その他の注意とコツは、以下のとおりです。

空白を入力するコツ

プログラムを見てもわかるとおり、C言語はすき間の多い言語です。行の間が1行空いていたり、「+」や「=」の記号の前後が空いていたり、いくつかの行が桁下げされていたりします。プログラムを初めて打ち込むとき、これを全部スペースキーで空ける人もありますが、それは不要。次のような要領で、すき間を作ってください。

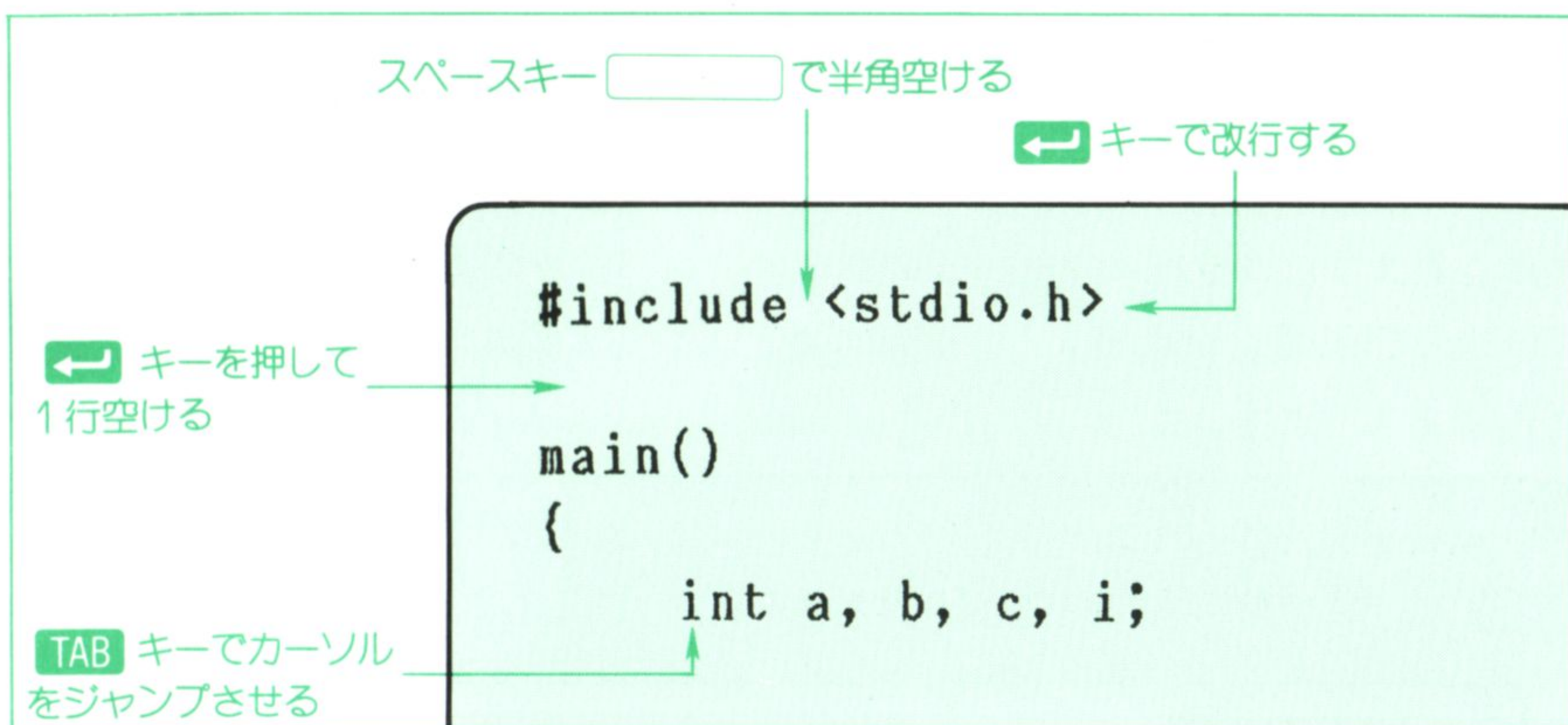
①1行の終わりは、↵キーを押して次の行に移る。
②行と行の間のすき間は、行の先頭で↵キーを押し、改行して1行空ける。
③文字と文字の空きは、半角のスペースを入れる(全角ではダメ)。
④行の先頭が4文字ぶんずつ桁下げされているものは、TABキーを利用して先頭を空ける(TABキーによるジャンプは、エディタによって4～8文字と幅がある。エディタの文字数に合わせるとよい)。

スペースキー [　　] で半角空ける

[↵] キーで改行する

[↵] キーを押して1行空ける

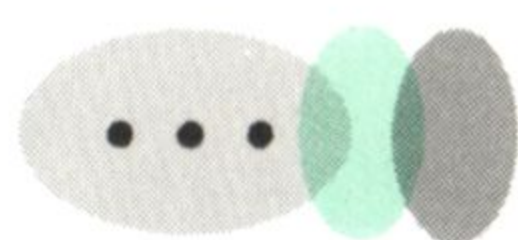
[TAB] キーでカーソルをジャンプさせる

```
#include <stdio.h>

main()
{
	int a, b, c, i;
```

6 日本語の入力

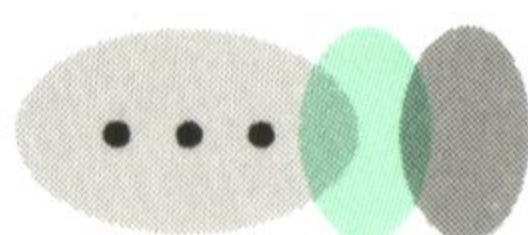
途中に日本語の文字が入っています。使っているコンパイラなどで日本語入力ができない場合は、英語またはローマ字に置きかえてください。サンプルプログラムに書かれている日本語は、画面に表示するメッセージですから、これらが英語またはローマ字で表示されるだけです。日本語を入力するときの注意は、以下のとおりです。

①日本語を入力するときは、 CTRL ＋ XFER キーを押し、フロントプロセッサを起動する。

②日本語の前後の記号「”」や「()」などは、フロントプロセッサを解除したのち、半角文字で入れる。

③すき間のスペースを、まちがって全角にしない。ただし「”」ではさまれた内側は、全角のスペースでもよい。

プログラムの保存

「Quick C」では、編集画面にプログラムを打ち込んだら、 GRPH ＋R→Sキーを押すだけでプログラムが実行できます。ただし、念のために打ち込んだプログラムをディスクに保存しておきましょう。 GRPH ＋Fキーでファイルメニューを呼び出し、「保存」を選びます。

エディタが付属していないコンパイラの場合は、エディタを使ってプログラムを打ち込んだら、それをまず保存してから、プログラムをコンパイル→リンク→実行します。どちらの場合も、プログラムの名前は、たとえば「prog3-1.c」などのように、拡張子「c」を必ずつけておきます。

エラーが出たときの処置

プログラムにエラーがあるときは、コンパイルを中止してエラーメッセージが表示されます。エラーメッセージの形式はコンパイラによってさまざまですが、「Quick C」の場合は次のような表示のあと、カーソルがプログラムのエラー箇所付近に止まります。

```
error C2143: (1/1)
';'が、'if'の前に現れなければなりません.
```

▲エラーメッセージの例

```
   c = 0
■  if ( a <= b )
↑      for ( i = a; i <= b; i++ )
カーソル      c = c + i;
```

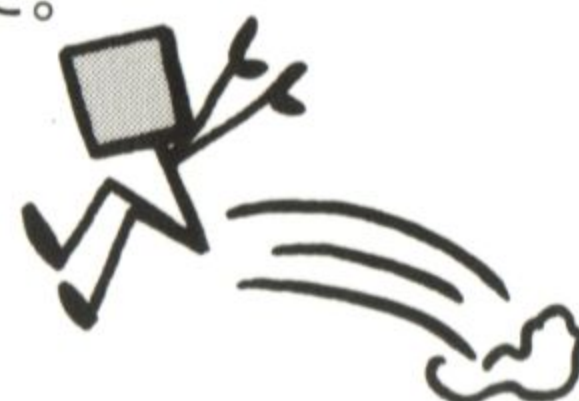
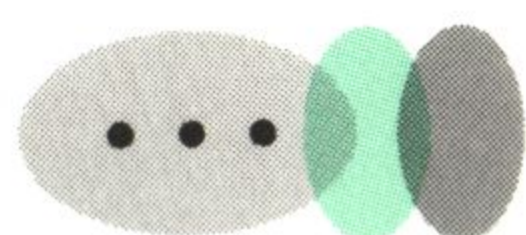
▲エラーの箇所

この例は「c=0」の次に「;」(セミコロン) を忘れたために出たエラーです。カーソルの指し示す位置は、エラーがある行そのものではなく、直前の行に原因のあることが多いので、前後を注意深く調べてまちがいがないかどうかを点検し、エラー箇所を修正して再び実行してください。

コラム

エラーファイル

「MIFES-98」や「RED2」「Final」など市販のエディタの多くは、エラーチェックに便利なタグジャンプという機能を持っている。本文で述べたように、カーソルがプログラムのエラー箇所にジャンプする機能がこれ。しくみは、バッチ・ファイルでコンパイル時にあらかじめエラーファイルを作るように指定しておくと (p.56参照)、エラーメッセージが出たときに自動的にエラーファイルが作られる。そのエラーファイルがエディタに呼び出され、エラーメッセージに対応するプログラムの行へ、カーソルをワンタッチでジャンプさせるというものだ。

b:err

プログラムの実行例

コンパイル→リンクの手順が正しく行われると、いよいよプログラムの実行です。「Quick C」の場合は、前述したようにメニュー画面から「R/実行」を選ぶだけです（GRPH＋R→S）。画面が「Quick C」から MS-DOS（エムエス・ドス）のレベルに戻り、ここでプログラムが実行されます。

このプログラムを実行すると、まず整数の入力を促すメッセージが表示されます。そこで、たとえば「2,35」と入力すると、2から35までの和が計算されて答えが表示されます。

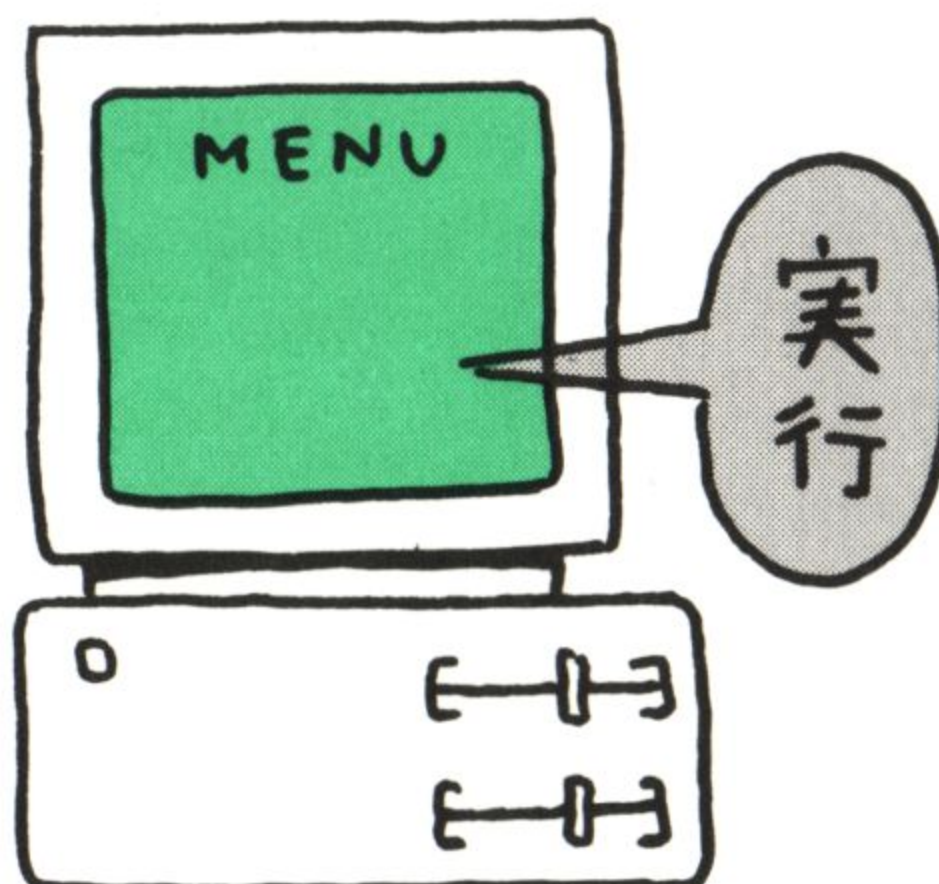
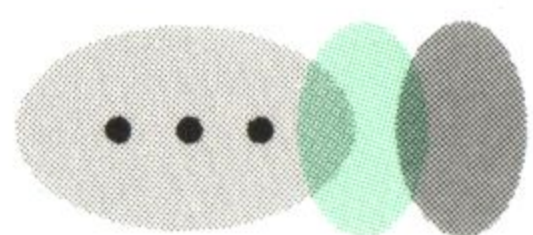
```
整数を２つ入力してください ←――入力を促すメッセージ
例  1,100
2,35 ↵                    ←――入力する
2から35までの和を計算します
答え   : 629              ←――実行のメッセージと結果
```

▲サンプルプログラムの実行結果

プログラムを実行するのは、リンクによって作られた、拡張子「EXE」の実行ファイルです。その前のコンパイルの段階では、オブジェクト・ファイルも作られていますが、統合開発環境を持つ「Quick C」などはメモリ上で実行までを行うため、それらのファイルが作られません。「Quick C」を終了して MS-DOS のレベルに戻り、DIR（ディレクトリ）コマンドでディスクの中を探しても、「PROG3-1.OBJ」と「PROG3-1.EXE」はありません。

統合開発環境以外のコンパイラでは、コンパイル→リンクが正しく終了したら、「PROG3-1.EXE」というファイルができていることを確認したあと、次のように入力してプログラムを実行させる必要があります。

☞A＞PROG3-1 ↵

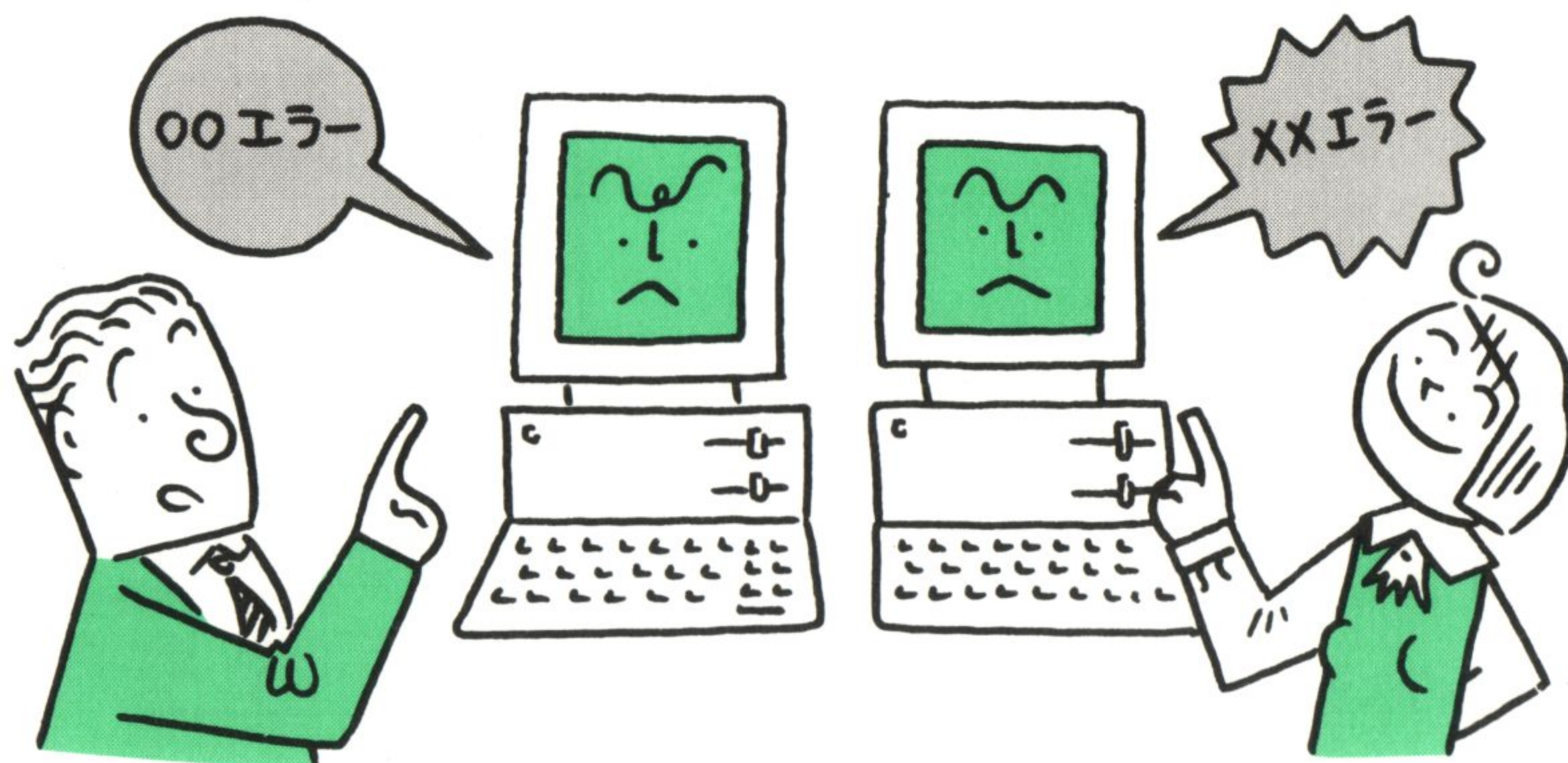
統合開発環境のコンパイラ

その他のコンパイラ

プログラミングとエラー

実際に例題がうまく実行できたでしょうか？　慣れるまではエラーばかり出て先へ進まないかもしれませんが、うまくいかなくとも気にやむことはありません。初めからエラーのないプログラムを書ける人は、だれ一人としていないのですから。サンプルのようなプログラム例があるときは、自分で最初からプログラミングするのに比べて、まだ楽です。例題と自分の打ち込んだプログラムがどう違うかを調べれば、まちがいが簡単にわかるからです。

どのようなまちがいのときに、どのようなエラーが出るかを覚えていくようにしてください。そのためにも(？)、エラーを何度も出すことが勉強になります。慣れてくると「このエラーメッセージは、こういうときに出るんだな」ということがわかるようになるものです。

「Quick C」は関数名のまちがいまでチェックしてくれますが、それ以外のコンパイラは普通、そこまではチェックしません。たとえば「printf」を「print」と書きまちがえたような場合、コンパイルはそのまま終了し、リンクの段階でエラーと判断され、実行できなくなってしまうのです。

エラーメッセージを何度も出していると、だんだんカンのようなものが身についてきて、メッセージを見てプログラムを直すのが快感になってきます。こうなるとしめたものです。

プログラム例を打ち込むだけでなく、今後、自分でプログラムを作るときは、わけのわからないエラーがたくさん出ることでしょう。そのとき、エラーメッセージに慣れているのといないのとでは、エラーチェックのスピードが大きく違ってきます。

プログラム記述の原則

プログラムの体裁

6 プログラム中の空白

C言語のプログラムはすき間が多い、とすでに説明しました。行間が1行ぶん空いていたり、行によっては4～8文字ぶん先頭位置から桁下げされていたりすることです。ただし、こういった書きかたはC言語自体の決まりではありません。その証拠に、次のように空白部分をすべて詰めてもプログラムの意味は変わらず、スペースを空けたときと同じ働きをします。

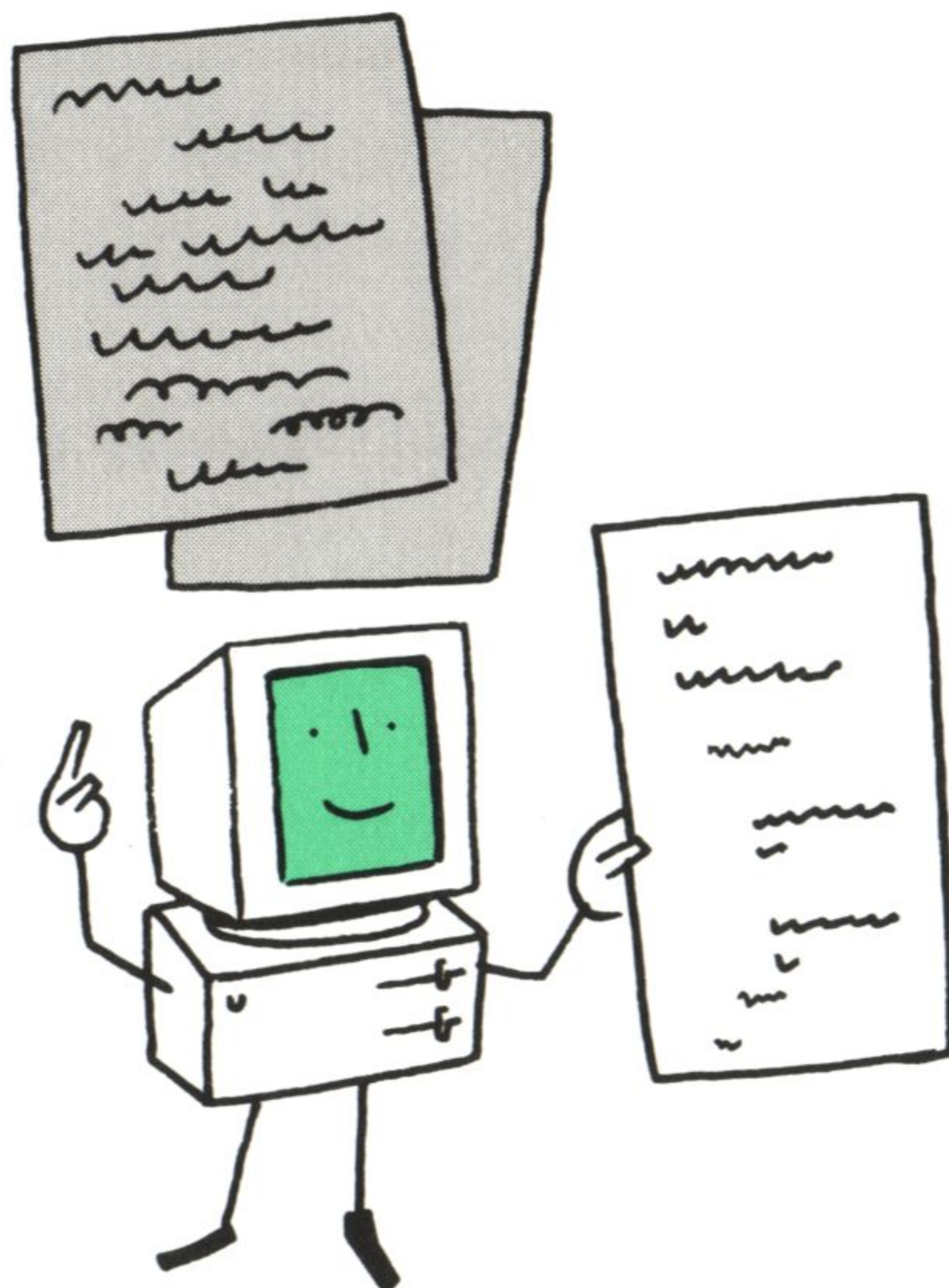
◎詰めて記述したサンプルプログラム

```
#include <stdio.h>
main(){int a,b,c,i;
printf("整数を2つ入力してください¥n例 1,100¥n");scanf("%d,%d",&a,&b);
printf("%dから%dまでの和を計算します¥n",a,b);
c = 0;if(a<=b) for(i=a;i<=b;i++)c=c+i;else for(i=b;i<=a;i++) c=c+i;
printf("答え  : %d¥n",c);}
```

実は、C言語のプログラムは、どこにどれだけ空白を設けてもよいのです。ところが、あまり詰めて書かれたプログラムは読みにくいし、何がなにやらわけがわかりません。そこで、C言語を作った人たちが、プログラムを見やすくするために、次のようなプログラムの書きかたを提唱しました。現在のほとんどのプログラマも、これらの書式に従っています。

①1行に1文だけしか書かない。

②プログラムは桁を下げて書く。

③プログラムの意味のまとまりごとに、行を空けて書く。

6コメントの挿入

わかりやすいプログラムのために、もうひとつ大事なことがあります。それは、プログラムの中に注釈文をたくさん入れることです。

/＊ ………… ＊/

このように「/＊」と「＊/」ではさまれた部分は「コメント」と呼ばれ、プログラムに関する注意書きや注釈文を書くところです。サンプルプログラムにコメントを入れてみましょう。コメントは、プログラムの実行には影響しません。

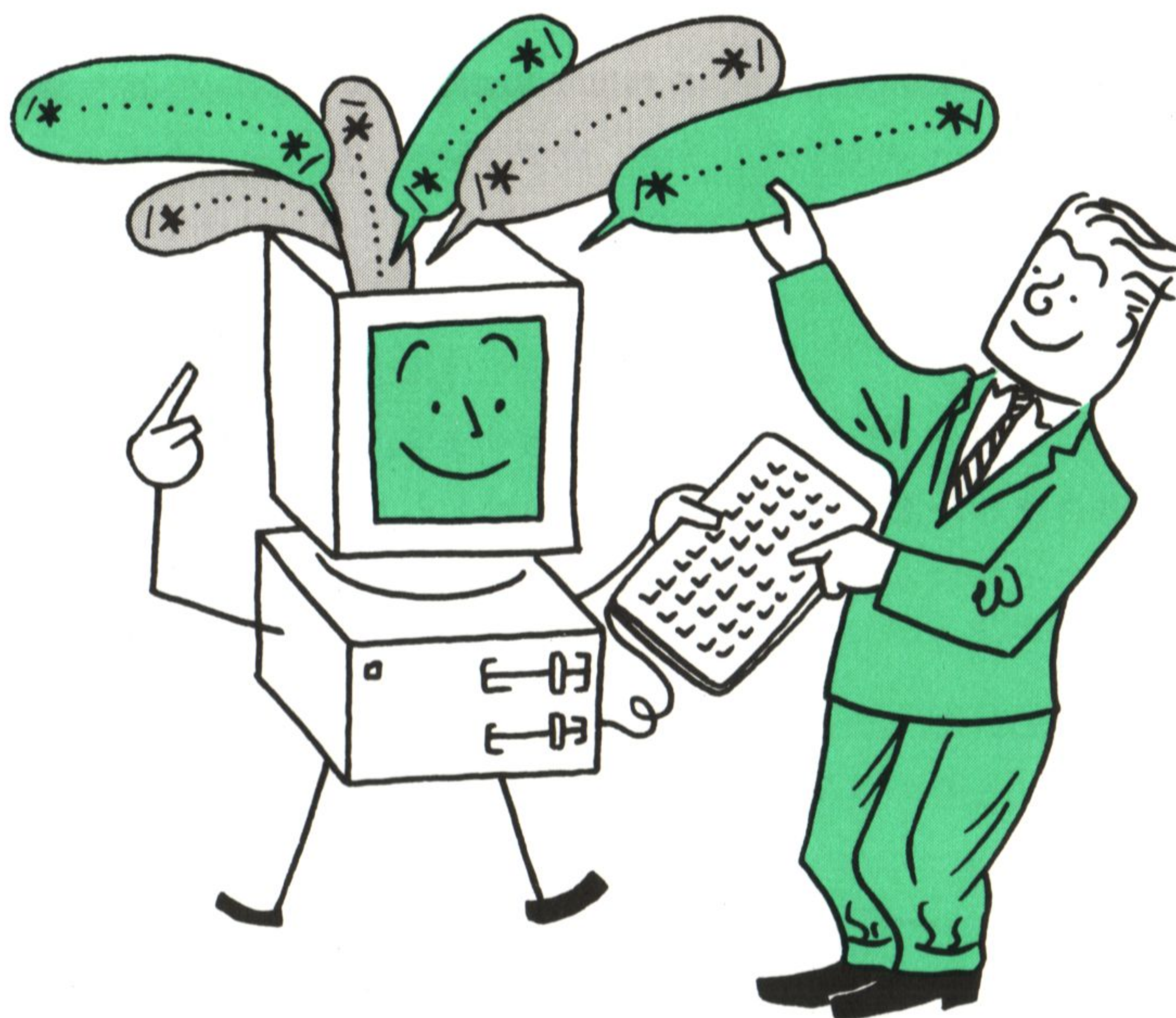
◉コメントを入れたサンプルプログラム

```
#include <stdio.h>           /* ヘッダファイルを指定する */

main()
{
    int a, b, c, i;          /* 変数を宣言する */

    printf("整数を２つ入力してください¥n例 1,100¥n");
                             /* メッセージを表示する */

    scanf("%d,%d", &a, &b); /* a,bの値をキーボードから入力 */

    printf("%dから%dまでの和を計算します¥n", a, b);
                             /* メッセージを表示する */

    c = 0;                   /* 変数cに0を代入する */
    if ( a <= b )            /* aからbまでの和を計算 */
        for ( i = a; i <= b; i++ )
            c = c + i;

    else                     /* bからaまでの和を計算 */
        for ( i = b; i <= a; i++ )
            c = c + i;

    printf("答え  : %d¥n", c);  /* 答えを表示する */
}
```

それぞれの具体的な意味については次項で解説しますが、コメントを入れることでプログラムが一見してわかりやすくなったと感じられるでしょう。ただ、コメントを入れるのは手間がかかります。「自分が作ったんだから忘れないさ」と、コメントをまったく入れない人もたくさんいます。たしかに、作ったばかりのときはプログラムの内容を十分把握していても、しばらく時間がたつと、どういう

意味のプログラムだったか忘れてしまうものです。

また、プログラムは自分だけのものではありません。別の人たちが見たり、作り直したりすることがしばしばあります。田川さんの例のように「プログラムを作った人が退社して内容を聞くわけにいかない」というケースもよくある話です。そういうとき、コメントはプログラムの内容を解く鍵になります。コメントはできるだけ多めに入れるようにしましょう。

プログラムの空白とコメント、これは2つともプログラムにとって、とても大事なものなのです。

6 プログラムは「main( )」から始まる

サンプルプログラムには、最初の行に「# include <stdio.h>」という文があります。この文は「ヘッダファイルを指定する」という意味なのですが、この話は少しむずかしいので、プログラム解説の最後で説明します。

さて、次の行に、

```
main()
```

という文字が見えます。これはC言語には必ずついている決まり文句で、どんなプログラムにも必ずあり、メインと呼びます。英語では、主となるものをメイン、

副となるものをサブといいます。「main( )」は、まさにプログラムで主となるものです。そして、C 言語プログラムは「main( )」から始まります。

C 言語以外の一般の言語では、プログラムをその働きによって、

- メインルーチン
- サブルーチン

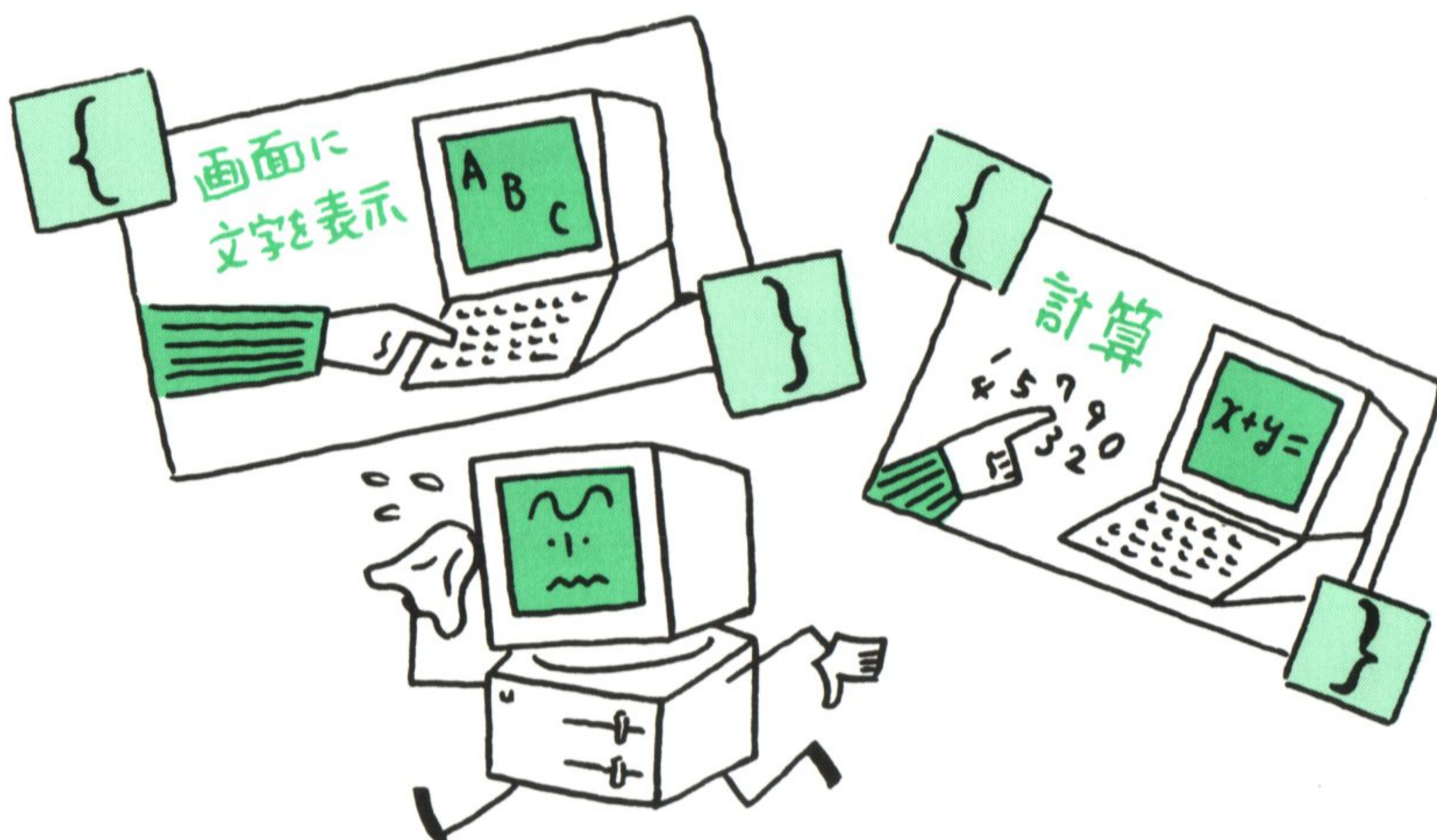
というように呼び分けます。C 言語では、サブルーチンの代わりに関数というものを使いますので、この関係は次のように置きかえられます。

- メイン関数（これが main( )）
- 関数

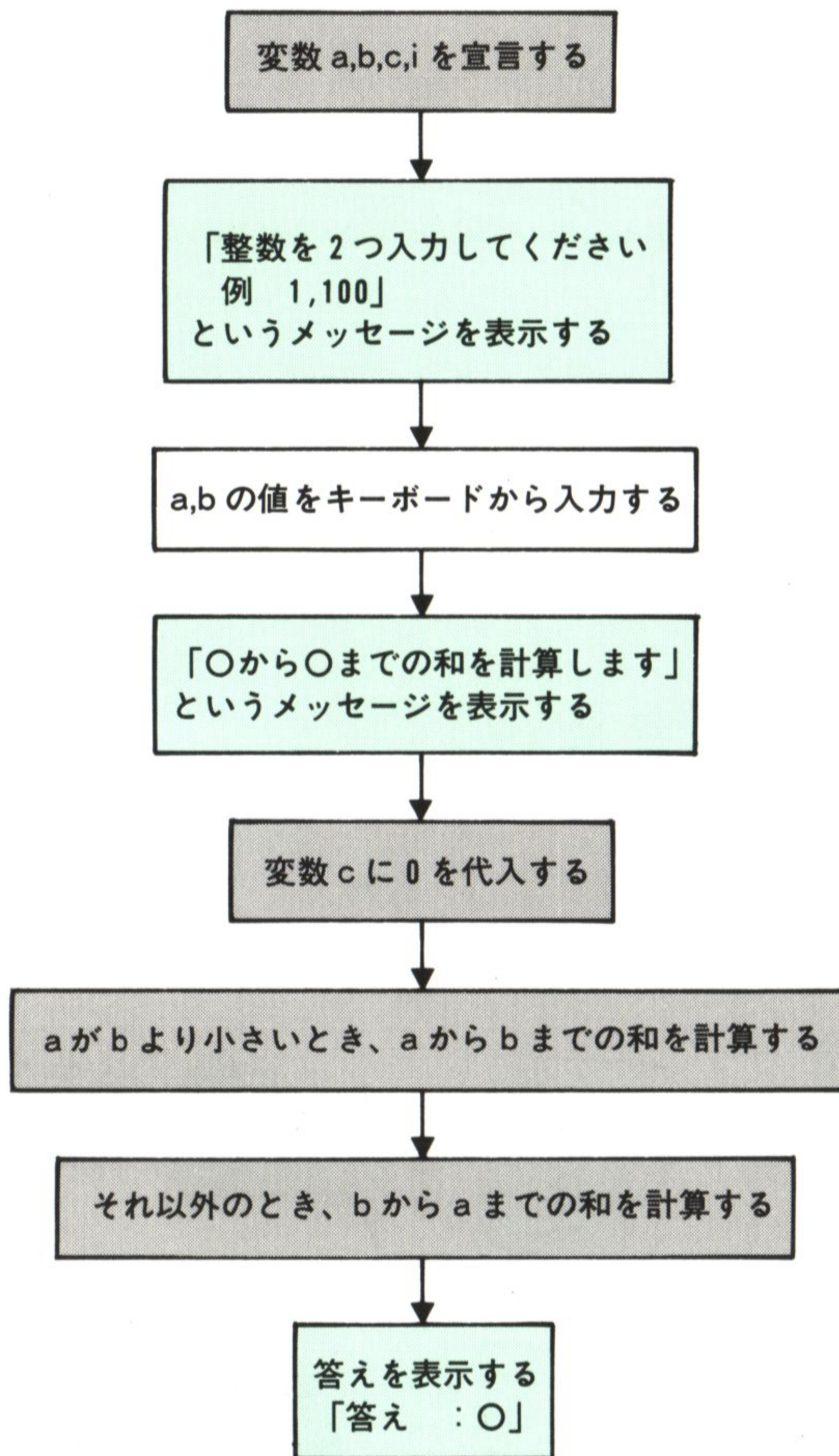
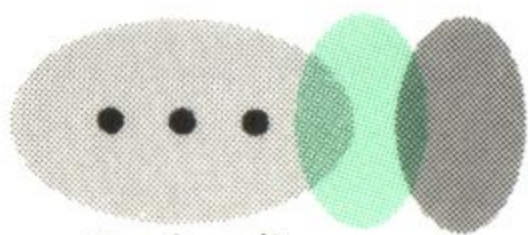
関数とは、複数の値を与えると、その答えや結果を返すもので、C 言語のひとつの単位です。「z＝f(x,y,w,v)」のような、数学の関数と同じようなものだと思ってよいでしょう。関数については、順を追って説明していきます。

6 「{　}」はプログラムのひとかたまり

「main( )」の次の行は、「{ 」が書かれているだけです。そしてずっと下の行を見ると、最後にこのカッコを閉じる「 } 」があります。

「{　}」ではさまれた部分は、プログラムのひとかたまり（ブロックのようなもの）です。1つのブロックでプログラムを構成することもありますし、複数のブロックで1つのプログラムにすることもあります。最初の「 { 」がなかったり、最後の「 } 」を忘れたりすると、プログラムがどこから始まってどこで終わるのかわかりません。このようなプログラムは、エラーになるのです。

キーワードの解説

ではいよいよ、プログラムの解読を始めましょう。コメント入りのプログラムで示したように、サンプルプログラムは、次のような流れになっています。

変数 a,b,c,i を宣言する
↓
「整数を 2 つ入力してください
例　1,100」
というメッセージを表示する
↓
a,b の値をキーボードから入力する
↓
「○から○までの和を計算します」
というメッセージを表示する
↓
変数 c に 0 を代入する
↓
a が b より小さいとき、a から b までの和を計算する
↓
それ以外のとき、b から a までの和を計算する
↓
答えを表示する
「答え　:○」

もう一度コメント入りのプログラムを見てみると、「printf」と書かれた行でメッセージを画面に表示し、「scanf」と書かれた行でキーボードからの入力を受けつけています。そして、a と b の値がどちらが大きいかによって、2 つの処理に分けられています。これらの文や処理が、C 言語のキーワードです。

ステートメント

BASICなどでは命令語のことをコマンドといいますが、C言語ではコマンドとはいわず、ステートメントと呼びます。意味は「文」ということですが、「文」という呼びかたは座りが悪いので、本書では以降、単独で出てくるときはステートメントと呼び、「○○文」のように前に何か言葉がつくときは「文」を使うことにします。

ステートメントには「宣言文」と「実行文」の2つの種類があります。

再度、サンプルプログラムに戻りましょう。

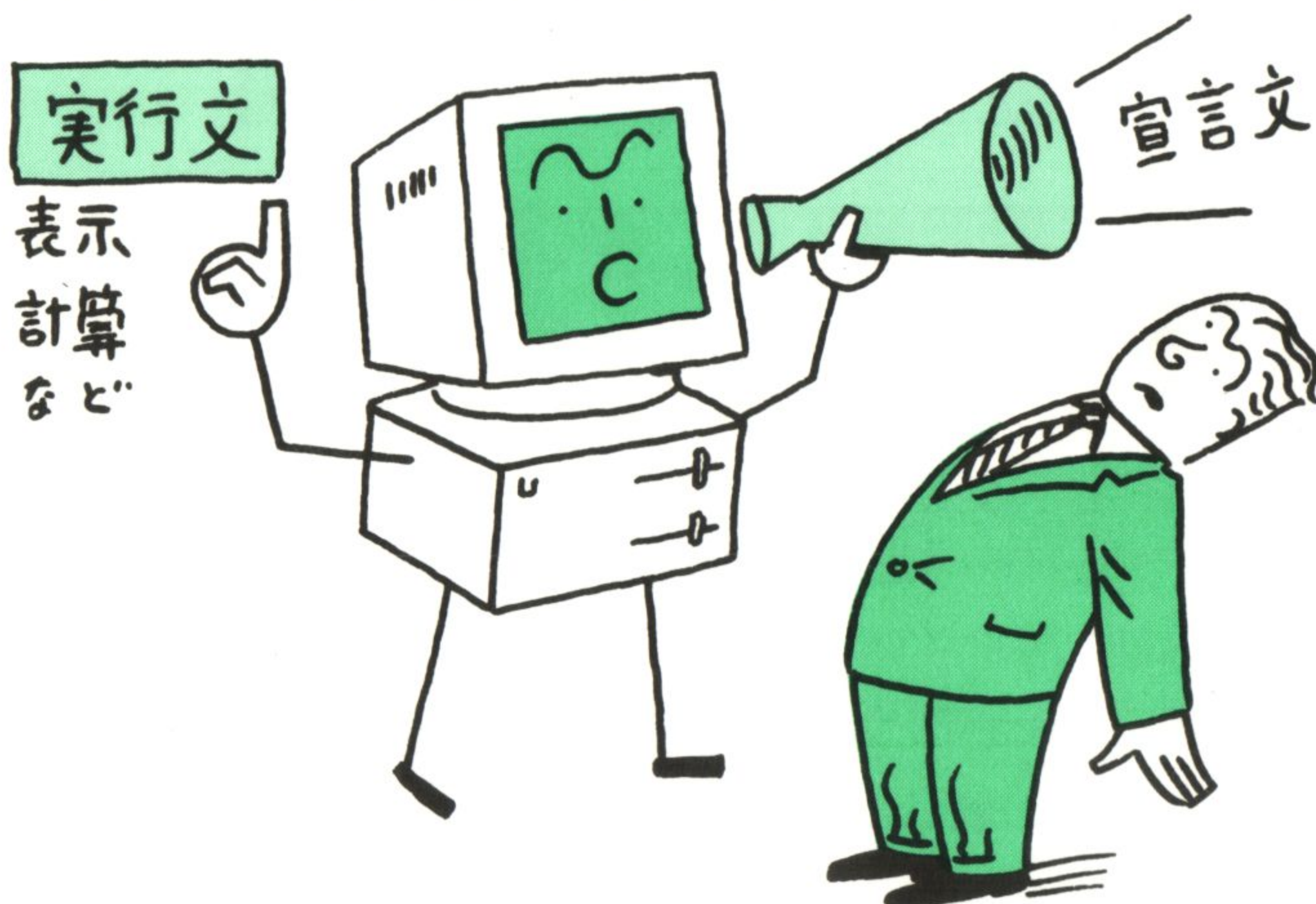
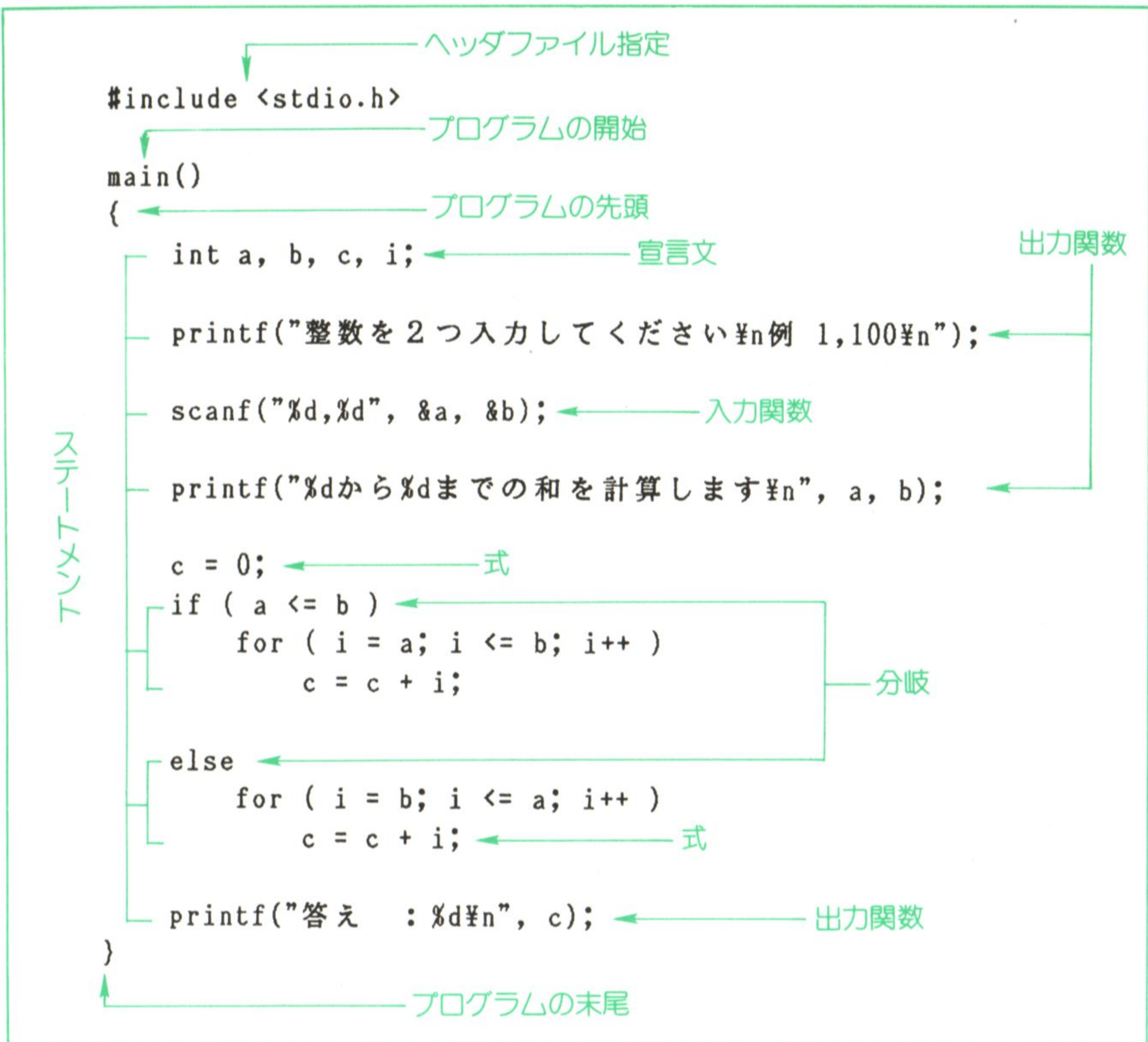
```
int a, b, c, i;

printf("整数を2つ入力してください¥n例 1,100¥n");
```

など、終わりに「;」(セミコロン)がつけられたものが、1つのステートメントです。

```
if ( a <= b )
    for ( i = a; i <= b; i++ )
        c = c + i;
```

のように3行に分かれていても、やはり最後の「;」までが1つのステートメントになります。その他の部分も同様です。

```
#include <stdio.h>
main()
{
    int a, b, c, i;

    printf("整数を２つ入力してください¥n例 1,100¥n");

    scanf("%d,%d", &a, &b);

    printf("%dから%dまでの和を計算します¥n", a, b);

    c = 0;
    if ( a <= b )
        for ( i = a; i <= b; i++ )
            c = c + i;

    else
        for ( i = b; i <= a; i++ )
            c = c + i;

    printf("答え　：%d¥n", c);
}
```

ヘッダファイル指定
プログラムの開始
プログラムの先頭
宣言文
出力関数
入力関数
ステートメント
式
分岐
式
出力関数
プログラムの末尾

▲サンプルプログラムのキーワード

変数を宣言する「宣言文」と変数

```
int a, b, c, i;
```

は宣言文というもので、「我輩は猫である」というように何かを宣言しています。我輩は猫である、の代わりに、この場合は、

a、b、c、i は整数である

という宣言を行っています。サンプルは「和を計算するプログラム」ですから、扱う数字のタイプを決めておく必要があるわけです。

この宣言は、コンピュータの約束によって「コンピュータの中に（正しくはメインメモリのどこかに）a、b、c、i という箱を用意して、以降、その中には整数だけを入れる」ということを意味しているのです。

この場合のa、b、cやiのことを、変数と呼びます。プログラムでは、「{　}」ではさまれたブロックの最初に、変数を宣言します。これは、必ずしも最初でなくてもよいのですが、その変数を使う前に宣言しておかなければなりません。宣言なしに変数を使うと、プログラムはエラーになります。

変数は、アルファベットと記号、およびそれに続く数字などで表現します。

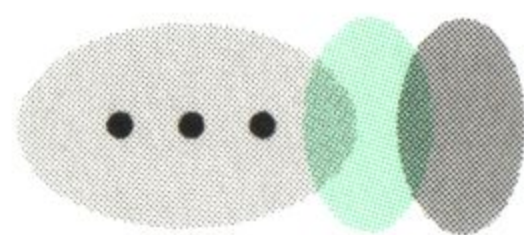
a1　b235　beatles

f_123　no _where_man

などのような変数はオーケーです。しかし、

123_f　help.me　thank-you

などは認められません。先頭が数字だったり、ピリオドのついたものは変数とはみなされないのです。また「-」は、ハイフンではなく、引き算のマイナス記号として扱われますから、変数には使えません。

変数名は、半角で最大31文字以内でつけてください。変数名の大文字と小文字は、別のものとして区別されます。日本語の漢字やカタカナ、ひらがなは、残念ながら使えません。

入力関数、出力関数

続いて、関数の説明に移りましょう。関数の代表は、**printf**(プリントエフ)文です。

```
printf("整数を２つ入力してください¥n例 1,100¥n");
```

これは画面に、次の２行を表示する働きをします。

```
整数を２つ入力してください
例　1,100
```

「printf」などの関数は、厳密にいえばＣ言語そのものではありません。Ｃ言語にあらかじめ用意されているもので、これらを標準(入出力)関数といいます。「printf」はそのうちの出力関数のひとつです。文字を画面に表示したり、キーボードからの入力を受けつけたりするとき、Ｃ言語ではすべてこれらの標準関数を使って行います。

次の行にある「**scanf**(スキャンエフ)」が、標準関数のうちの入力関数と呼ばれるもので、このステートメントには、aとbの箱にキーボードから入力した値を入れよ、という意味があります。

「printf」と「scanf」に関して、サンプルプログラムにおける働きをまとめておきましょう。

```
printf("整数を2つ入力してください¥n例 1,100¥n");
```

画面に次の2行を表示する

```
整数を2つ入力してください
例 1,100
```

```
scanf("%d,%d", &a, &b);
```

キーボードから入力した数字を、変数aと変数bの箱に入れる

```
printf("%dから%dまでの和を計算します¥n", a, b);
```

画面に「○から○までの和を計算します」と表示する
○にはaとbの整数の値が入る

```
printf("答え  :%d¥n", c);
```

画面に「答え :○」と表示する
○にはcの整数の値が入る

「printf」「scanf」などの関数については、項を改めてくわしく説明します。ここでは、それぞれがどういう働きをするのかだけでもつかんでおいてください。

コラム

関数の話

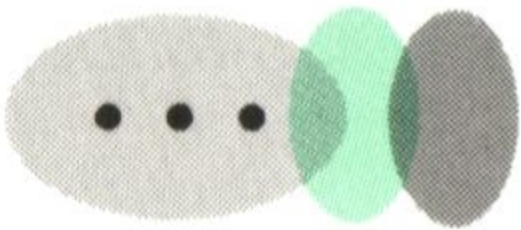
一般的に関数は、数学で用いられるのと同様に、

z = f (x, y, w, v)

のような形をしていなければならない。x、y、w、vに値を入れると、zに1つの値が返される。ところがサンプルプログラムでは、どの関数もこのような形にはなっていない。

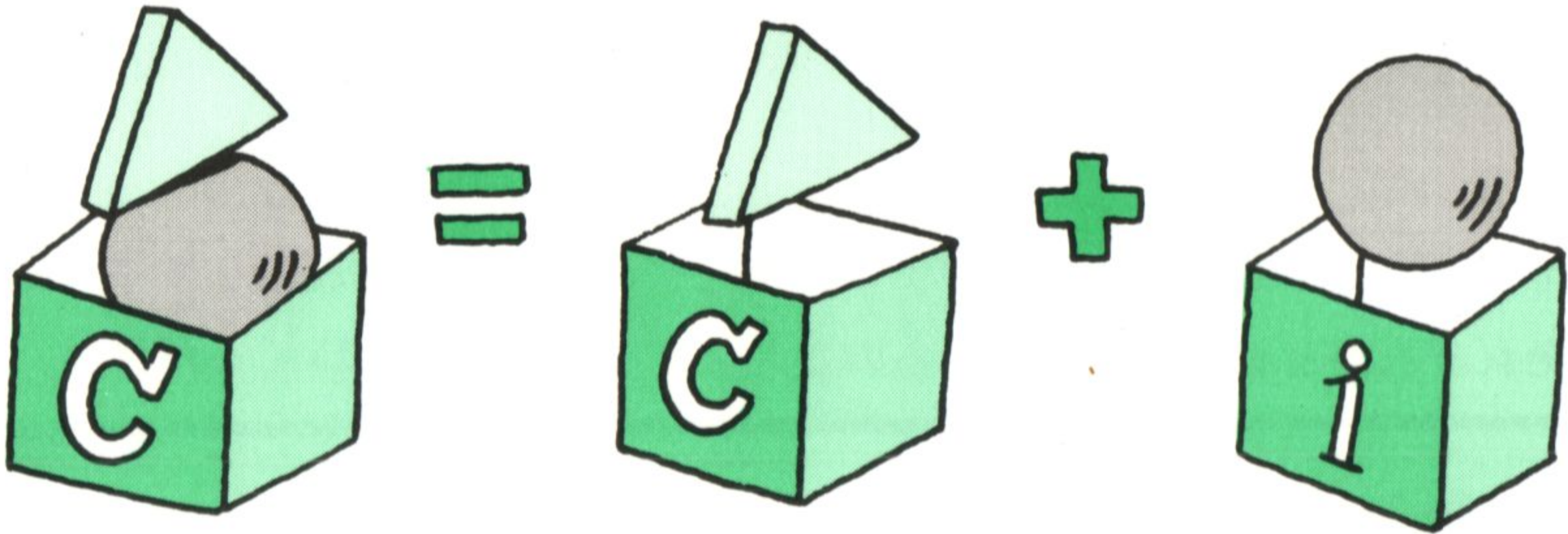
実は、「printf」なども本来は、

c = printf (" ……",……) ;

のような形をしていなければならないのである。ただ、「printf」「scanf」といった標準関数の場合、返される値が当面は必要なく、働きだけに意味があるので、「c=」の部分が省略されているわけだ。

式と演算子

プログラムの中で計算式を設定するとき、左辺には必ず変数を置くのが決まりになっています。たとえば、次のようです（これらの式もステートメントだから、最後に「；」がついているのに注意）。

```
c = 0;
c = c + i;
```

1行目の「c＝0；」は、変数cに0を入れるという意味です。2行目の、「c＝c＋i；」は、何かちょっとおかしな感じがします。数学ではありえない式ですが、コンピュータの世界ではこの書きかたがしばしば出てきます。これは、
①箱cの中身と、箱iの中身を足して、
②その結果を箱cに入れる、
という意味です。つまり「＝」には、右辺の結果を左辺に代入するという意味があるのです。

右辺の計算式は、数学と似たように表現します。コンピュータで扱われる四則演算には、下表の記号が用いられます。

演算の種類	数学の演算記号	コンピュータの演算子
足し算	＋	＋
引き算	－	－
掛け算	×	＊
割り算	÷	/

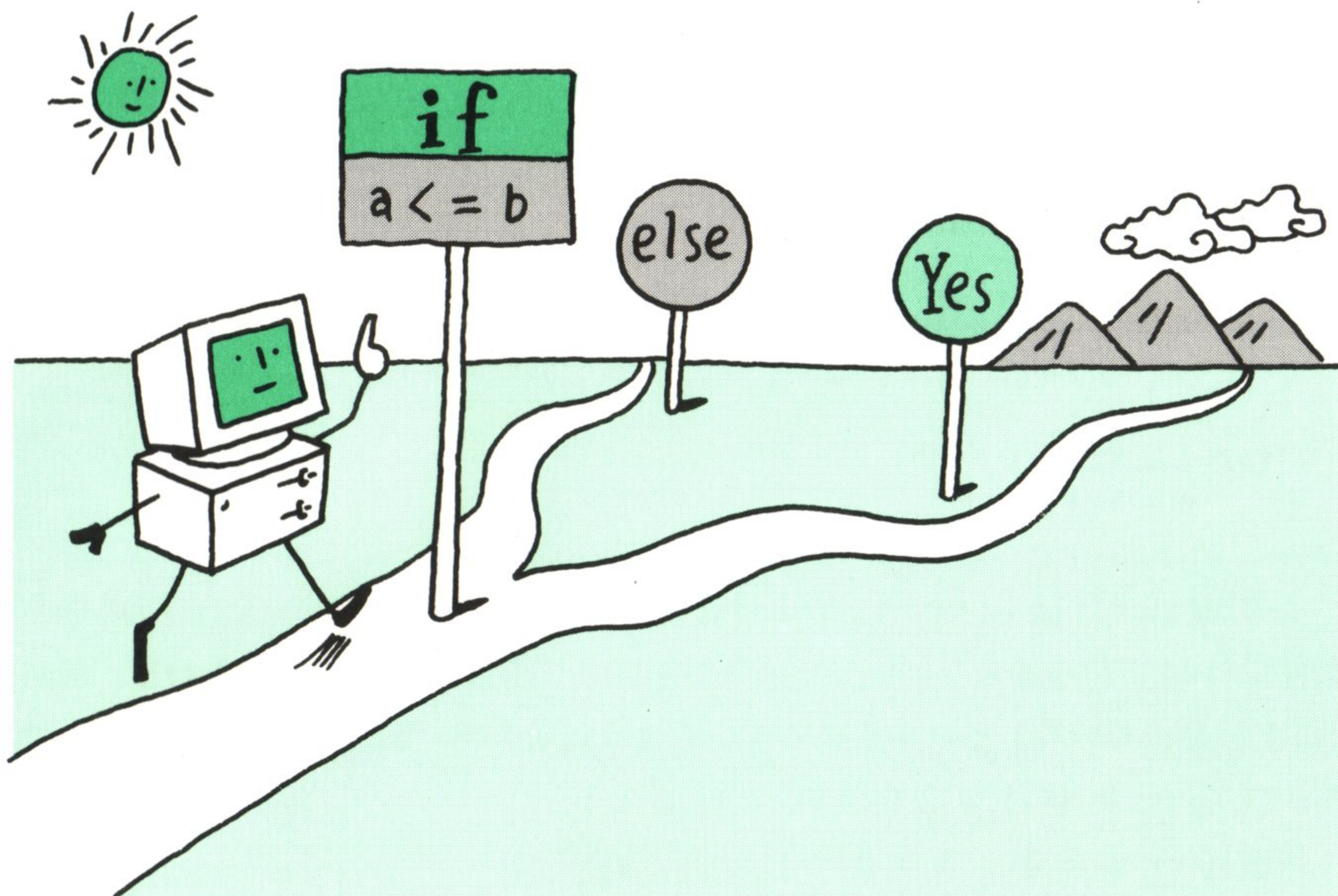
このうち、キーボードには、「＋」「－」はありますが、「×」と「÷」は見あたりません。その代わりに、「＊」「／」を使用します。これらの記号を演算子と呼びます。

分岐

分かれ道で右か左かというとき、つまり与えられた条件によって処理を変えるときは、if（イフ）文というものを使います。基本的な書式は、次のとおりです。

```
if (……)          ← 条件
   ………;          ←┐
else                ├ 条件によってどちらかの処理に飛ぶ
   …………;         ←┘
```

右か左かを選ぶときの判断基準は、「if」に続く「(……)」の部分に記入します。ここに書かれた条件で、右か左かが決まるのです。このプログラムでは、条件「a <= b」によって「if」以降の行に進むか、「else」以降の行に進むかが分かれます。そして、このような処理を分岐と呼びます。

```
if ( a <= b )              /* aからbまでの和を計算 */
    for ( i = a; i <= b; i++ )
        c = c + i;

else                       /* bからaまでの和を計算 */
    for ( i = b; i <= a; i++ )
            c = c + i;
```

つまり、整数aが整数bより小さいときは、「if」の次の行へ進み、

a　a+1　a+2　…… b

までをすべて足していきます。

逆にaがbより大きければ、「else」に続く行へ進み、

b　b+1　b+2　…… a

までをすべて足していきます。

このように分岐処理では、aとbのどちらが小さいかなど、条件をあらかじめ判断して、処理を変えるようにしておかなければなりません。そしてこれらは、コンピュータが自動的に判断できないため、プログラムで正確に指示しておく必要があります。

ループ

同じところを何度もぐるぐるまわるような繰り返し処理を、ループと呼びます。ループとは輪のことで、東京の山手線のような環状線もループです。基本的な書式は、次のようになります。

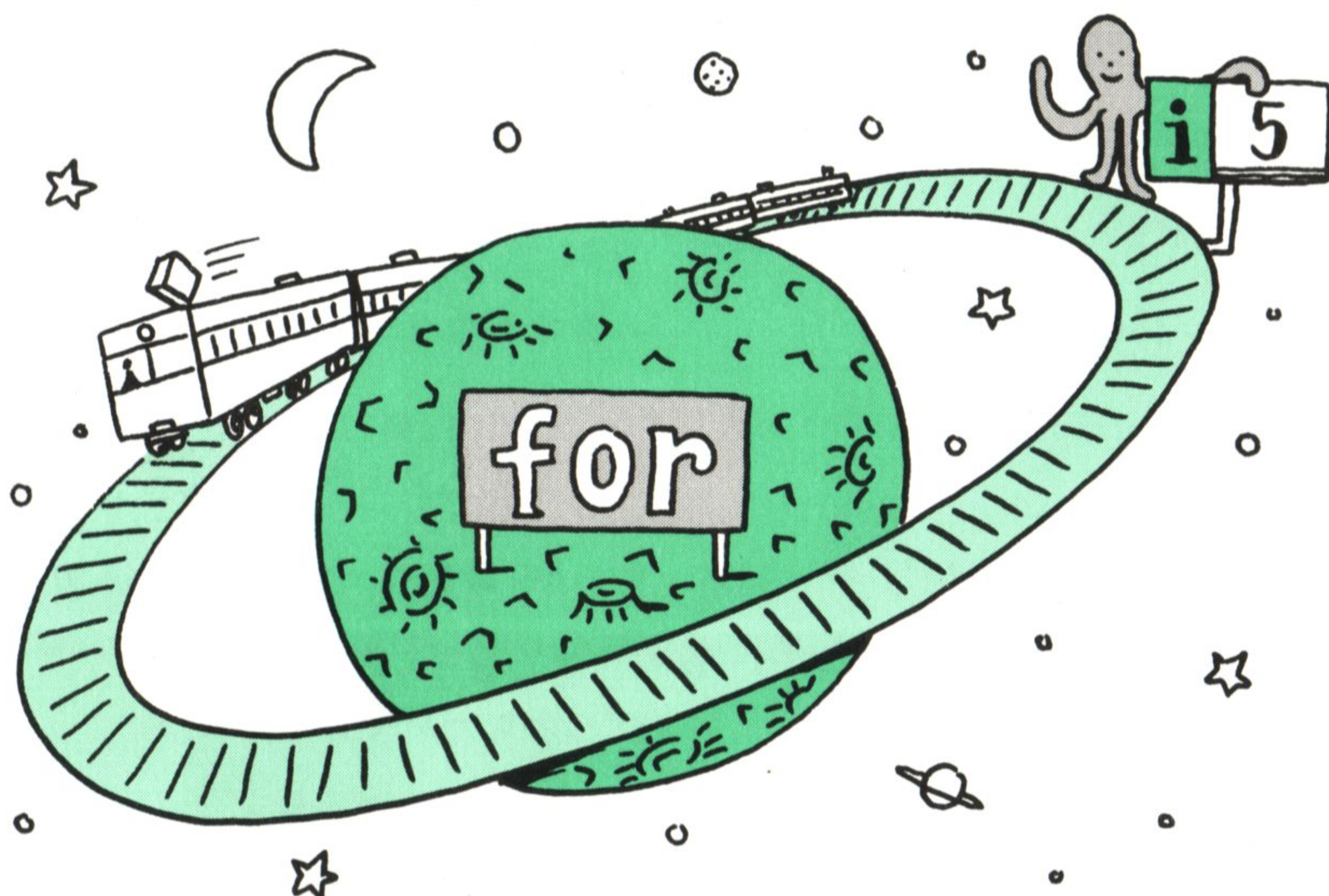
```
for (……) {
    ………    ←――――この部分を何度も繰り返す
}
```

これは「(……)」に書かれた条件を満たしているあいだは、「{　}」の中の処理を何度も繰り返すものです。プログラム例では「{　}」がなく「for (……)」に続く文が1行に書かれています。繰り返しの部分が1行だけの場合「{　}」はつけませんが、2行以上になる場合は必ず「{　}」ではさみます。

サンプルプログラムのこの部分を、解読してみましょう。まず、ループの前の「c=0;」という式で、cの最初の値（初期値）が0に設定されます。

```
for ( i = a; i <= b; i++ )
    c = c + i;
```

この**for**（フォー）ループは「c=c+i;」の1行を何度も繰り返す処理です。「(　)」の中の条件は「iはaから始まり、bより小さいか同じになるまでループがまわり、毎回iに1を足していく」という意味です。たとえばaが3、bが5の場合、1回目のループではc = 0、i = 3なので、c = c + i; は、

1回目……c = 0 + 3 = 3

になります。これでcに3が入ります。2回目のループではc＝3、i＝4なので、

　2回目……c＝3＋4＝7

同じく3回目はc＝7、i＝5なので、

　3回目……c＝7＋5＝12

となります。これでiがbの値である5になったので、ループが終了します。

　次のループはbがaより小さい場合で、iがbからaになるまで同じように足し算を繰り返します。

　C言語のループには、**for**（フォー）ループのほかに、

　while（ホワイル）ループ

　do～while（ドゥ・ホワイル）ループ

の2つがありますが、指定した条件のあいだぐるぐるまわって同じ処理を繰り返すところはすべて同じです。

キーワードの実行中の働き

　以上で、キーワードの意味がわかりました。サンプルプログラムをもう一度動かして、内容を順に確認してみましょう。説明のため行の左に番号をつけましたが、実際のC言語のプログラムには行番号がつけられることはありません。

◉サンプルプログラムの実行過程

```
 1:  #include <stdio.h>          /* ヘッダファイルを指定する */
 2:
 3:  main()
 4:  {
 5:      int a, b, c, i;         /* 変数を宣言する */
 6:
 7:      printf("整数を2つ入力してください¥n例 1,100¥n");
 8:                              /* メッセージを表示する */
 9:
10:      scanf("%d,%d", &a, &b); /* a,bの値をキーボードから入力 */
11:
12:      printf("%dから%dまでの和を計算します¥n", a, b);
13:                              /* メッセージを表示する */
14:
15:      c = 0;                  /* 変数cに0を代入する */
16:          if ( a <= b )       /* aからbまでの和を計算 */
17:              for ( i = a; i <= b; i++ )
18:                  c = c + i;
19:
20:      else                    /* bからaまでの和を計算 */
21:          for ( i = b; i <= a; i++ )
22:              c = c + i;
23:
24:      printf("答え  : %d¥n", c);  /* 答えを表示する */
25:
26:  }
```

❶プログラムの開始と、入力を促すメッセージの表示

プログラムを実行すると、行4までで開始が認識され、まず行7の**printf**(プリントエフ)文でメッセージが2行表示されます。

```
整数を2つ入力してください
例 1,100
```

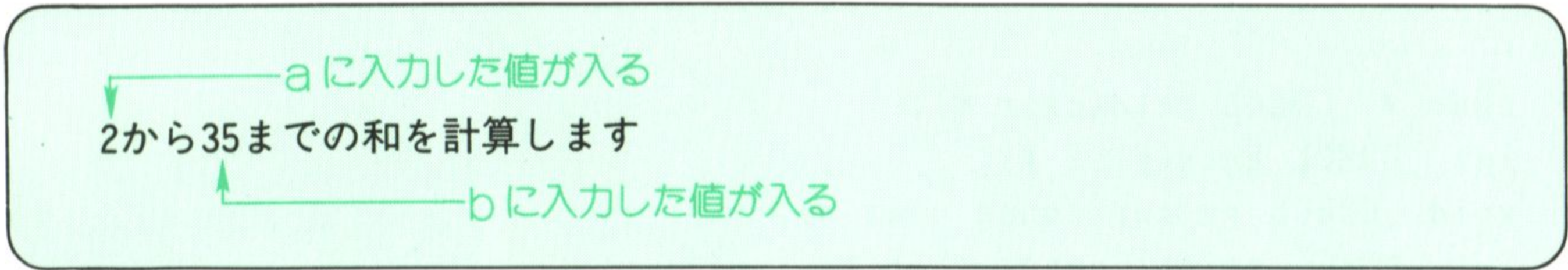
❷キーボードからの入力待ち

次に行10の**scanf**(スキャンエフ)文で、整数a、bの入力を待ちます。カーソルが点滅しているはずですから、たとえば次のように入力して ↵ キーを押します。

☞**2, 35** ↵

❸入力された整数の表示

scanf 文で入力された値は行12の **printf** 文のメッセージ中に代入され、それが表示されます。

```
 ┌────────a に入力した値が入る
 ↓
2から35までの和を計算します
     ↑
     └────────b に入力した値が入る
```

❹変数 c の初期値を設定する

行15で、計算結果を入れる変数 c の初期値を 0 にしておきます。

❺和の計算

入力された条件によって、行16～18または行20～22で、和の計算を行います。2 は35より小さいので、この場合は行16～18で計算を行っています。もし「35, 2 」などと入力された場合は、行20～22に飛んで処理が行われます。

❻答え（結果）の表示と、プログラムの終了

変数 c の値が、計算結果です。それが行24の**printf** 文に代入され、 答えを示すメッセージが表示されます。

```
答え　：629
```

次の行26は終了を示す「}」ですから、これでプログラムが終了します。

プリプロセッサ――# include <stdio.h>

最後に、サンプルプログラムの行 1 にある「# include ＜stdio.h＞」について説明しておきましょう。「#」のついた行はプリプロセッサと呼ばれ、プログラムをコンパイルする時点でプログラムにいろいろな指示を与える働きをします。

「# include　＜stdio.h＞」という**include**文は、「プログラムのこの行に stdio.h というファイルの内容を挿入する」という意味になります。

「stdio.h」は、標準入出力関数（printf、scanf、getchar、fprintfなど）に関する宣言の入っているファイルで、Standard input/outputの意味です。プログラムの中でこれらの関数を定められた機能で使用するときは、最初に、必ずこの**include**文を記述しておかなければなりません。

興味がある人は、エディタなどで「stdio.h」の内容を見てください。このファイルは通常、コンパイラの「¥include」というディレクトリに入っています。

なお、まちがって入力したりして、stdio.h の内容を書きかえないようにくれぐれも注意してください。

◉ stdio.h の内容の一部（Quick C）

```
…………
char * _CDECL gets(char *);
int _CDECL getw(FILE *);
void _CDECL perror(const char *);
int _CDECL printf(const char *, ...);
int _CDECL puts(const char *);
int _CDECL putw(int, FILE *);
int _CDECL remove(const char *);
int _CDECL rename(const char *, const char *);
void _CDECL rewind(FILE *);
int _CDECL rmtmp(void);
int _CDECL scanf(const char *, ...);
…………
```

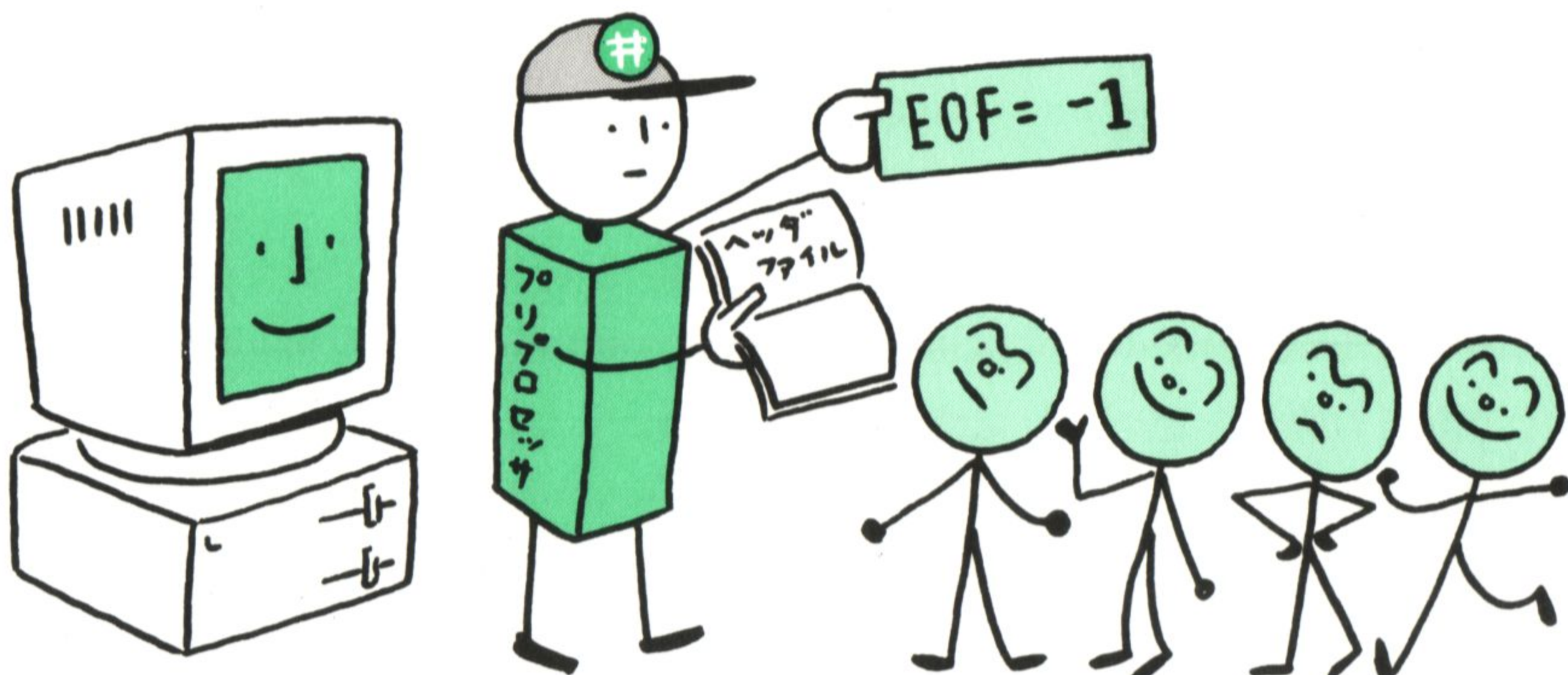
「stdio.h」のように、拡張子「H」がついたファイルは、ヘッダファイルと呼ばれています。ヘッダファイルは、プログラムに必要な宣言文や関数の型などが書かれているファイルです。

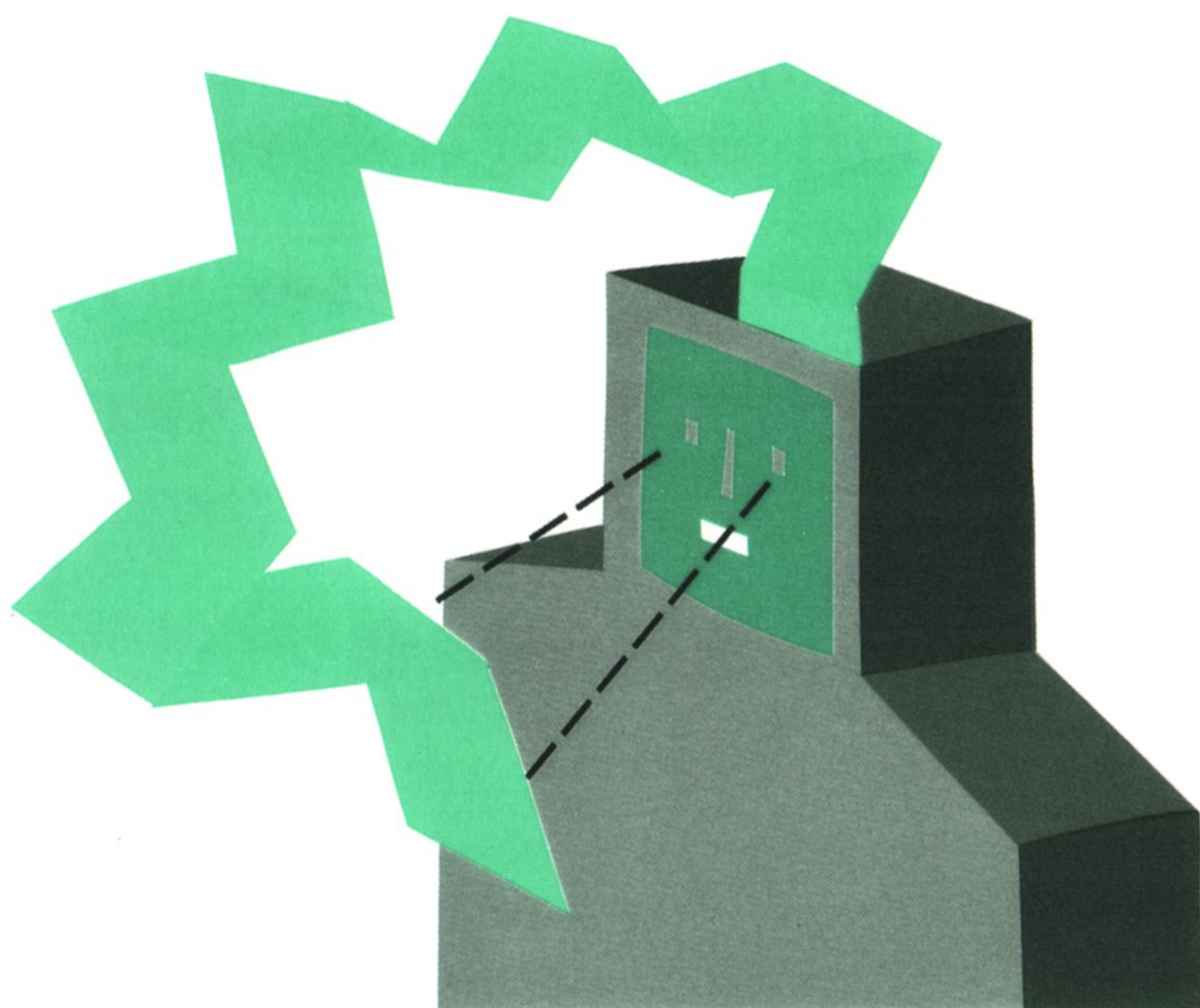
include（インクルード）文のほかに、よく使われるプリプロセッサには**define**（デファイン）文があり、

　#define　　名前　値

のように書かれます。

　#define　　EOF　－1

と書かれていると、それ以降のプログラム中にある「EOF」という文字は、つねに「－１」の値を持つという意味です。なお「EOF」は、決まった値をとる定数です。**define**（デファイン）文については、PART4 の定数のところで説明します。

PART 4

プログラミングのテクニック 1

田川部長、プログラミングのむずかしさを実感する。

玲子　ここまでで、プログラムのしくみがわかったかしら。

田川　いやぁ、むずかしいね。だいたいの流れはわかったけど、細かいところがまだよくわからないよ。特に「%、&、¥」など、へんな記号がたくさん出てくるのが気持ち悪くっていけない。

玲子　それらは、**printf**（プリントエフ）文や**scanf**（スキャンエフ）文につきものの記号なのよ。これまでは概略の説明だったので細かい記号の話まではしなかったけど、これからそういう説明に入っていくわね。その前に、PART3で説明したステートメントとか変数、分岐やループとかを、しっかり頭に入れておいてね。

田川　うーん、そのほかにもコメントを入れる話、いろいろな関数、足し算、引き算などの四則演算、式……いろいろあったから、忘れそうだね。

玲子　きょうはまず、C言語でプログラムを書くための決まり――これを文法といっています――について項目別に話をしましょう。PART3での概略をもとに、C言語の基本的な文法から勉強よ。

田川　いよいよ本格的か……しかし、私は昔から文法も暗記も苦手なんだなあ。

玲子　プログラミング言語の文法は、暗記じゃなくて、決まりを理解できればいいのよ。それにC言語は覚える言葉の数が少ないし、実際に使うときの例を見ながら説明するので、むずかしくはありません。その場で忘れてもらってもかまわないのよ。PART6ではプログラム例をいくつか紹介するつもりなので、それを見るときにこの章に戻って、わからない項目のところだけ復習するといいわね。

プログラムと変数

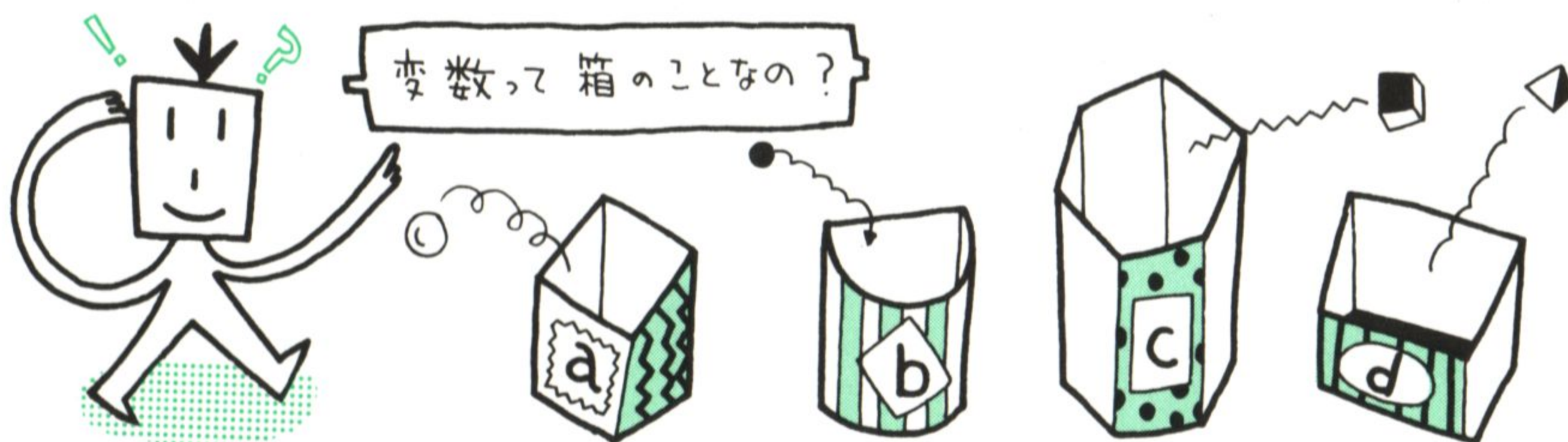
初めに変数ありき

PART3 のサンプルプログラムでは、最初に変数を宣言しました。

```
int a, b, c, i;
```

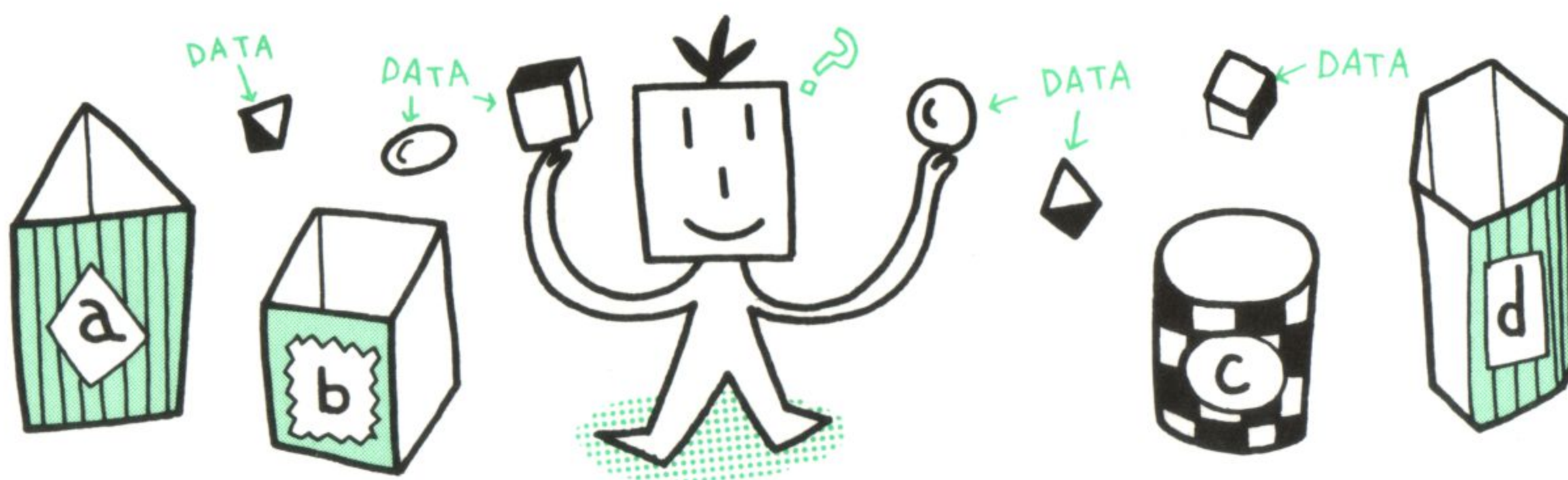
これは、変数 a、b、c、i が整数であるという宣言文でした。プログラムに使われる変数はすべて、使う前に宣言しておく必要があります。

変数はプログラムに欠かすことのできないもので、変数が理解できると、プログラミングの第一歩を踏み出したことになるといってよいでしょう。ひと口にいって、変数とはいろいろなものを入れる箱です。プログラミングの最初に箱をいくつも用意して、それぞれに名前をつけます。たとえば変数 a とは、a という名前をつけた箱のことです。

変数って箱のことなの？

6 変数の種類

C 言語で使われる箱には、大きいものから小さいものまで、大きく分けて下表の 4 種類があります。この、いろいろな大きさの箱に入れる中身のことをデータと呼びます。つまり、入れるデータの種類によって箱の型が異なるわけです。

◉変数の種類と大きさ

種　類	デ　ー　タ	大きさ	データ例
char（キャラクタ）型	文字データ	1バイト	A
int（インテジャ）型	整数データ	2バイト	10
float（フロート）型	小数点つきの数字データ	4バイト	3.14
double（ダブル）型	小数点つきの数字データ	8バイト	2.718281828

これらの箱をいくつも用意して、その中にデータを入れたり、ほかへ移したり、足したり引いたりするのがプログラムの働きです。

6 変数の働き

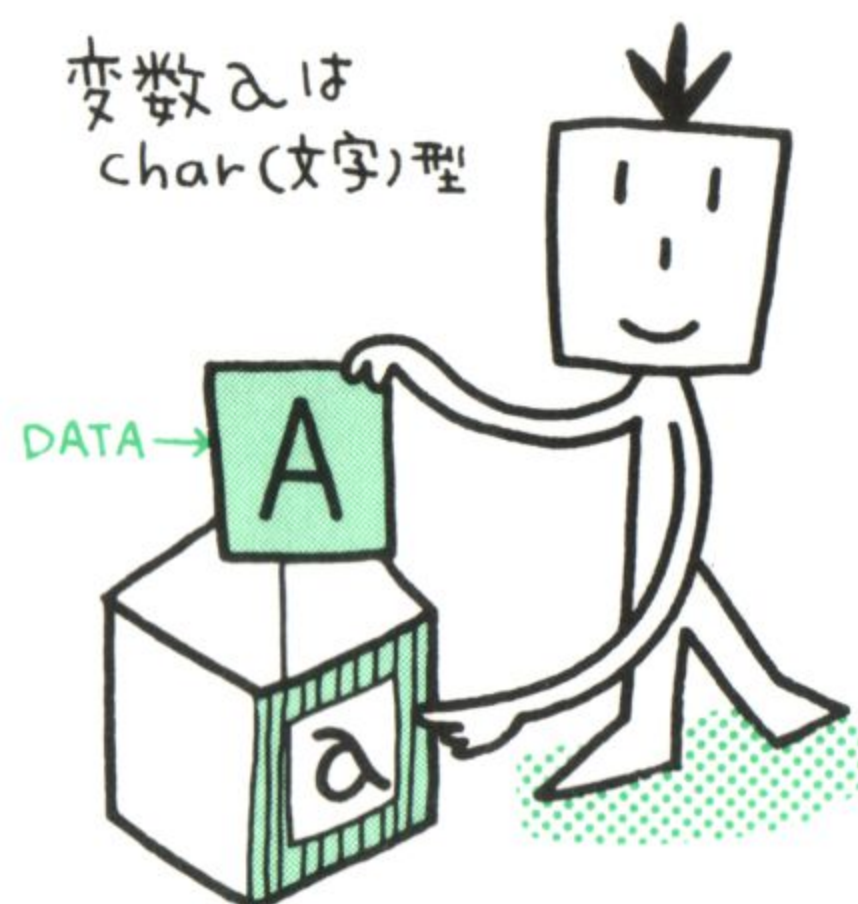
実際のプログラムの中で、変数の働きを考えてみましょう。以下のプログラムは、いろいろな箱に各種のデータを入れたものです。

◉変数にデータを入れるプログラム

```
main()
{
    char a;
    a = 'A';

    ……
    ……
}
```

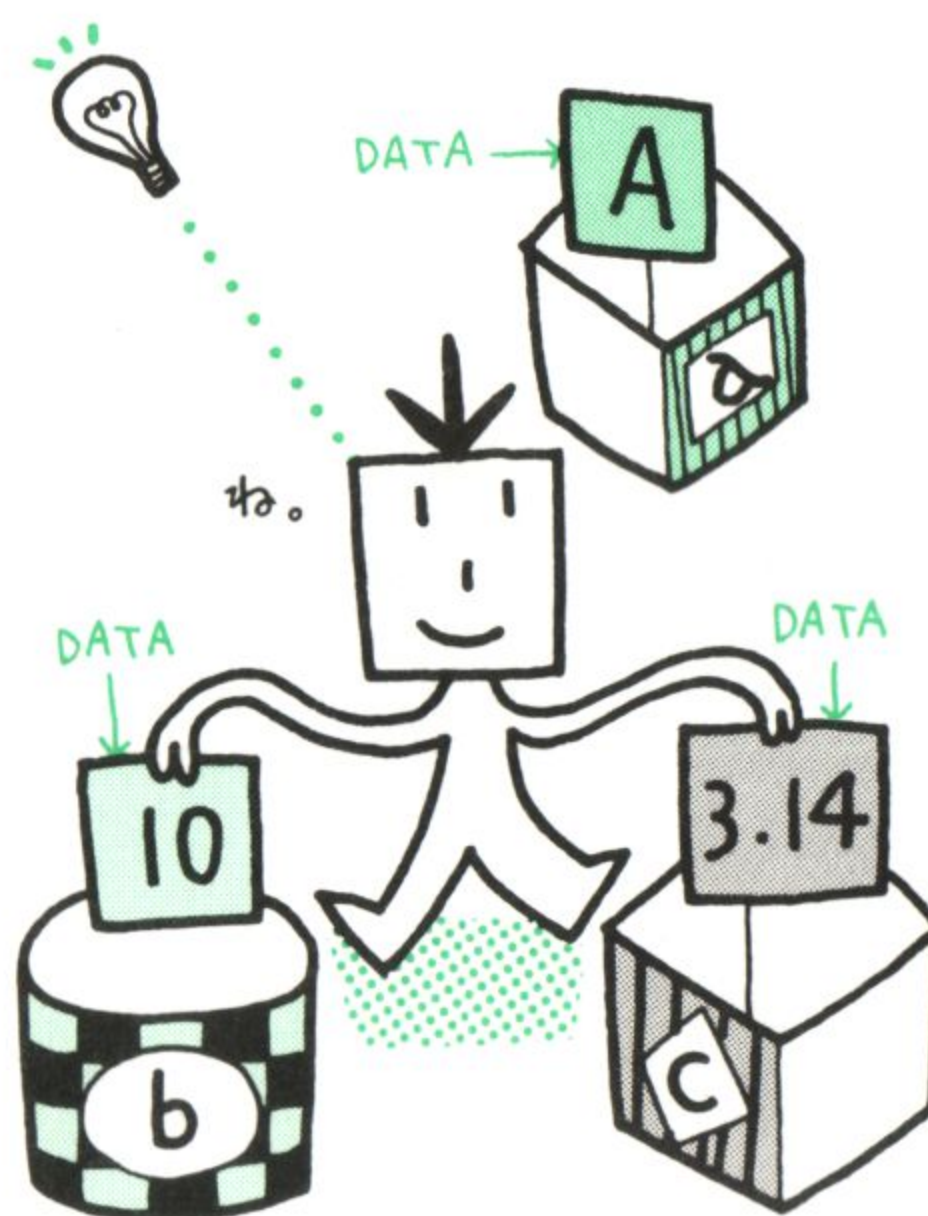
変数aを文字型に宣言

aにAを代入

```
main()
{
    char a;
    int b;
    float c;

    a = 'A';
    b = 10;
    c = 3.14;
    ……
    ……
}
```

変数a, b, cをそれぞれの型に宣言

変数aにAを代入

変数bに10を代入

変数cに3.14を代入

このように、変数に何かを入れるとき、プログラムでは「＝」（イコール）記号を使います。たとえば「a ＝ 'A';」の式は、a が A と同じだということではなく、右の A を左の変数 a に代入することを示しています。そして、文字を入れるときは「'A'」のように文字を「'」（シングルクォーテーション）で囲みます。最後に「；」（セミコロン）があるのは、前述したように、ステートメントを書くときの決まりです。

では、これらの箱はどこに用意されるのでしょうか。箱すなわち変数は、パソコン本体のメインメモリ、つまり RAM（ランダム・アクセス・メモリの略。読み書き両用メモリ）の中に置かれます。メインメモリには番地が割り振られていて、定められた位置に変数の箱が配置されます。

6 変数の大きさとバイト

4 種類の変数の型は、入れるデータの大きさが、1、2、4、8 バイトというように、倍々になっています。**char** 型の箱には、1 文字が入ります。この 1 文字ぶんの大きさを、コンピュータの単位で 1 バイトといいます。そして、1 バイトは、8 ビットのデータで構成されます。

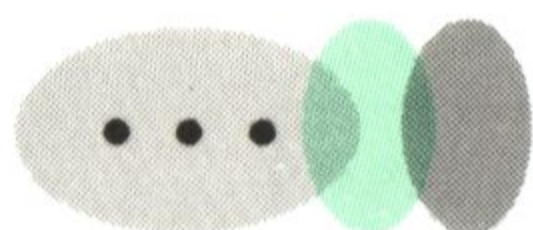
つまり、「1 バイト＝8ビット」です。ビットは16ビット・パソコンなどといういいかたでおなじみでしょう。この 1 バイトの中には、

01000001

などという 2 進数字が詰まっています。この 0 と 1 の集まりは、全体で「A」の文字を表しています。

int 型の箱は**char** 型の 2 倍、すなわち 2 バイトの大きさです。その中身は、

0011000000100110

などというように、16個の 2 進数字（この例は10進数にすると数字12326を表している）が入っています。ちなみに、2 バイトの大きさの箱に入れることのできる整数は、－32768～＋32767の間のものです。

変数の宣言

変数の宣言は、次のように書かれます。

char a; int b; float c; double d;	char 型の変数 a を宣言する int 型の変数 b を宣言する float 型の変数 c を宣言する double 型の変数 d を宣言する

同じ型の変数がいくつもあるときは、

```
int x;
int y;
int z;
```

と1行ずつ書いてもいいし、

```
int x, y, z;
```

とまとめることもできます。

変数は通常、次のように**main**(メイン)関数の「{　}」の中の先頭に書かれます。この位置に書いておくと、「main()」で始まる「{　}」(この、プログラムの一単位をブロックという)の中のどこででも使うことができます。

変数の初期値が決まっているときは、変数宣言と同時に、その値を入れることもできます。ですから、次のような書きかたも可能です。

◉変数の宣言と同時に初期値を入れる

```
main()
{
    char   a = 'A';
    int    b = 10;
    float  c = 3.14;

    ……
    ……
    ……


}
```

コラム

auto(自動)変数

「{　}」ではさまれたブロックの先頭に書かれた変数を、**auto**（オート）変数または自動変数と呼ぶ。宣言したブロック内なら、どの位置でも使うことができる。またプログラムの中にさらに小さいブロックがあり、その中で宣言された場合は、その小さいブロックの中だけで使うことができる。

たとえば下のプログラムでは、それぞれのブロックで整数型の変数 x が宣言されている。

変数にはこのほか、**static**（スタティック）（静的）変数、**extern**（エクスターナル）（外部）変数などがある。また、その変数がどの範囲で通用するかによって、グローバル（広域）変数、ローカル（狭域）変数という分けかたをされている（ふろく①変数について、を参照）。

ブロック大の変数宣言

```
main()
{
    int x;

    ……
    ……

    while() {

        int x;

        ……
        ……
    }

}
```

ブロック小の変数宣言

auto

static

変数が有効な範囲を決めるわけね。

auto

ブロック

static

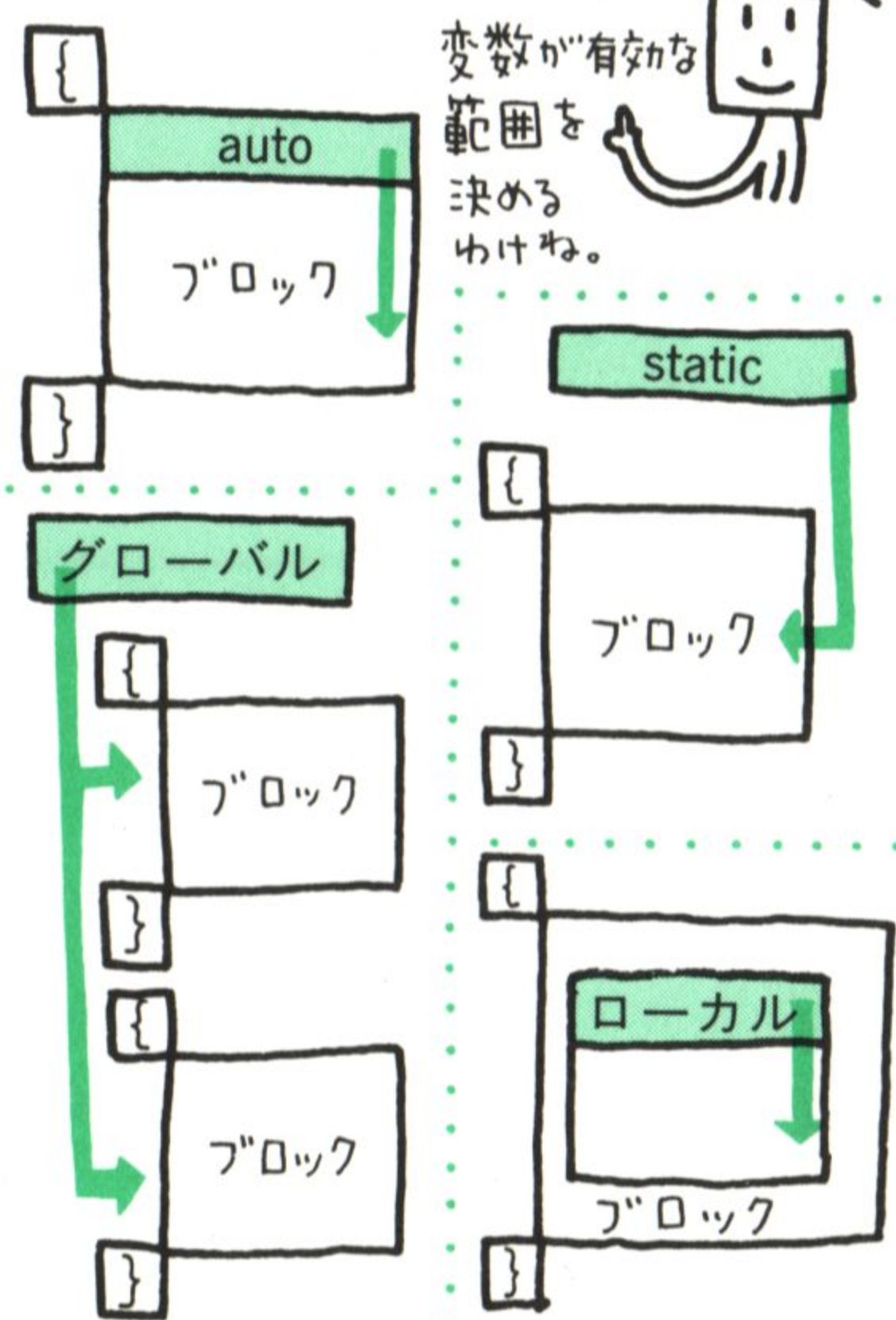
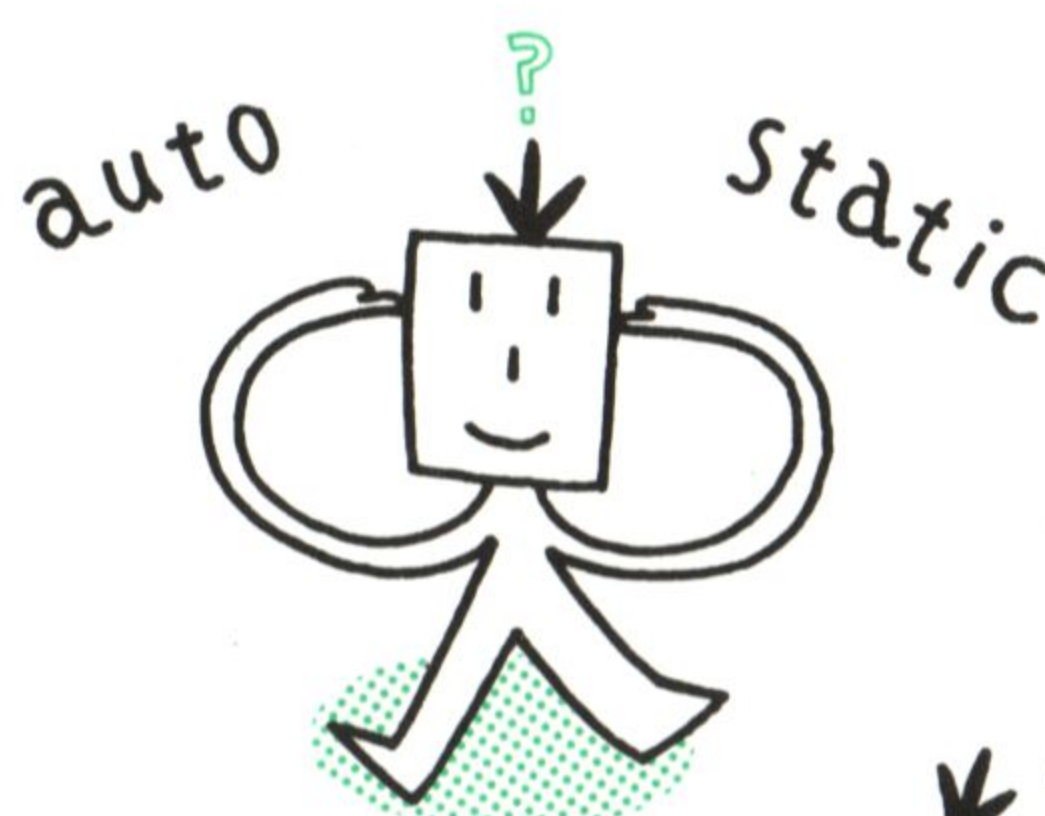
ブロック

グローバル

ブロック

ブロック

ローカル

ブロック

田川部長、変数に苦しむ。

玲子　変数は使う前に必ず宣言してください。**char**（キャラクタ）型、**int**（インテジャ）型、**float**（フロート）型、**double**（ダブル）型などのうちどの型かということは、とても重要なのよ。

田川　なんでこんなに変数の種類があるんだ⁉　変数宣言なんていちいちやらなくちゃいけないのかい、めんどうだなあ。

玲子　初心者のときは特にそう思うわね。BASIC（ベーシック）の場合には変数は使いたい場所で名前をつけて使うから「変数宣言」などする必要はないし、変数の型も気にしなくてよいのよ。

田川　そのほうが絶対簡単だよ。なんで変数の型なんかがあるんだい？

玲子　これは、コンピュータの本来の性質からきているの。どんなプログラミング言語の場合も、必ず変数の型というものはあるのよ。データを入れる箱が１種類だと簡単だけど、大きなデータを入れるためには大きめの箱をたくさん用意しなければいけませんね。ところが、箱を置く場所（＝メモリ）には制限があるから、使える箱の数は限られてしまいます。最小のメモリで最大の効率をあげるためには、箱を何種類か用意して、小さなデータは小さい箱に、大きなデータは大きい箱に入れるようにしたほうが効率がよいわけ。

　BASIC の場合だって、ユーザには見えないけれど、変数がいくつも用意されていて、パソコンのほうで適当に割り当ててくれるのよ。C 言語の場合は、プログラムで必ず宣言するという約束になっているわけなの。

配列と変数、定数

配列の役割

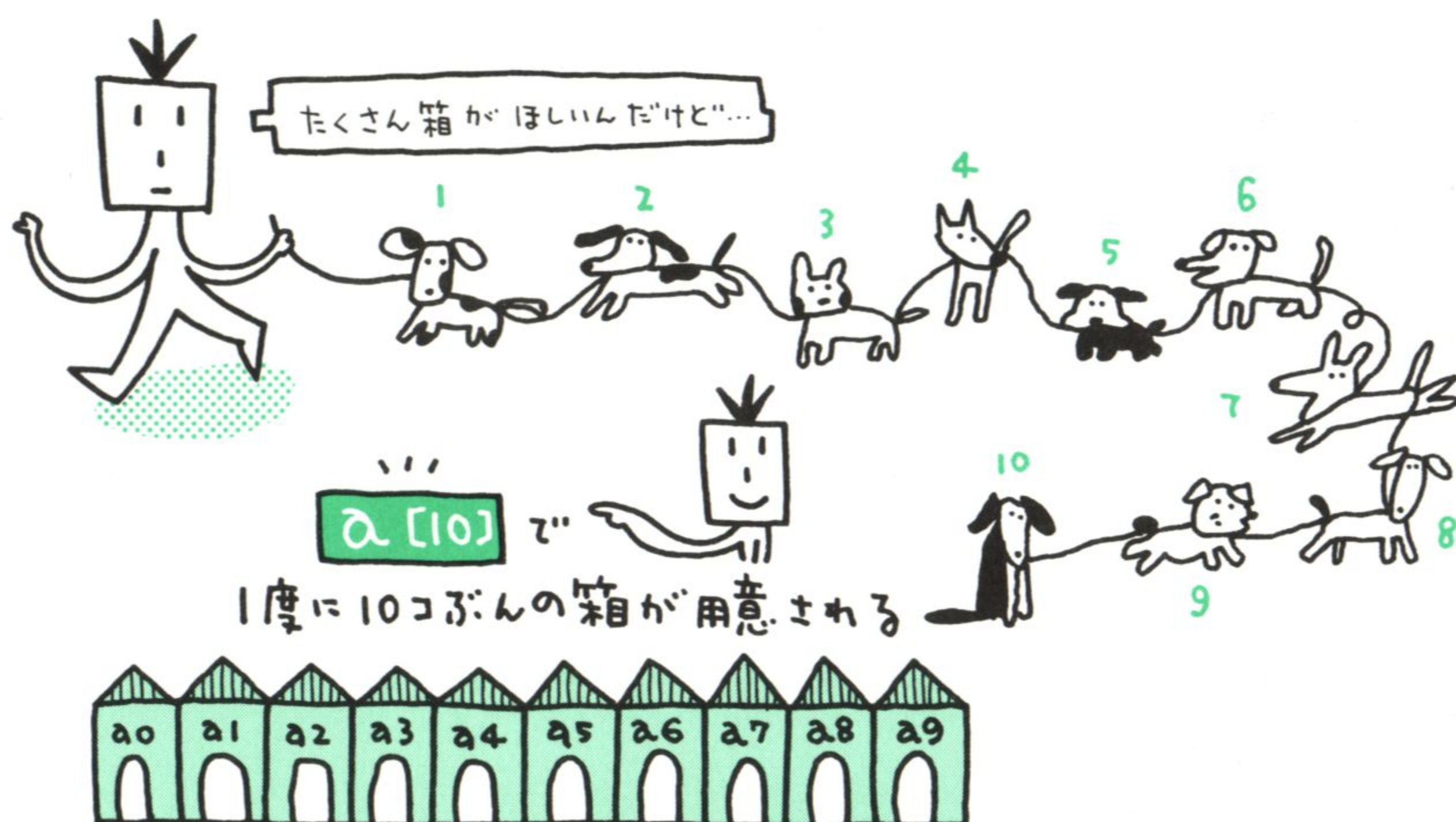
変数を使うとき、変数名1個につき箱が1個割り当てられました。ところが、まとめて10個とか20個といったデータを扱うとき、その数だけの箱を用意しなければなりません。宣言にも、それだけの手間がかかります。そこで、同じタイプの変数をまとめて扱うために、配列というものが用いられます。配列は、プログラムの中で、たとえば次のように宣言します。

```
int a[10];
```

こうすると、配列a用として、箱が10個ぶん用意されるわけです。配列で用意した箱のそれぞれは、

a[0]　a[1]　a[2]　a[3]……

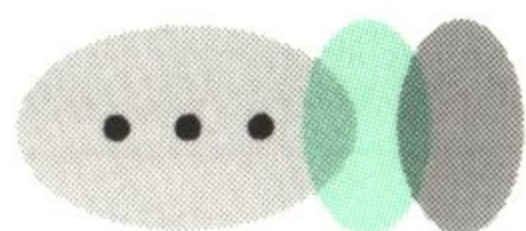
というように書かれます。この1つひとつを配列要素と呼び、たとえば10個の場合は、a[0]〜a[9]になります。a[1]〜a[10]ではないので注意してください。

この配列に、1から10までの数字を入れてみましょう。配列にデータを入れるときも、変数と同じように1つずつ「＝」で数字を代入していきます。

◉配列に文字を入れる

```
main()
{
    int a[10];  ←――――― 配列を10個宣言

    a[0] = 1;
    a[1] = 2;
    a[2] = 3;
    a[3] = 4;
    a[4] = 5;
    a[5] = 6;
    a[6] = 7;
    a[7] = 8;
    a[8] = 9;
    a[9] = 10;

    ……
    ……

}
```

この例を見ると、配列はめんどうなだけでなんのメリットもない気がするかもしれません。ところが、このような場合、ループ機能と組み合わせるとプログラムが簡単になります。次の例は、配列 i が 0 ～ 9 まで順に変わるあいだ、「a[i]＝i+1」という計算を行うループです。

◉配列をループで利用する

```
main()
{
    int a[10], i;

    for (i = 0; i < 10; i++)    ┐
        a[i] = i+1;             ┘―― ループ
    ……
    ……
}
```

文字データを取り扱う配列

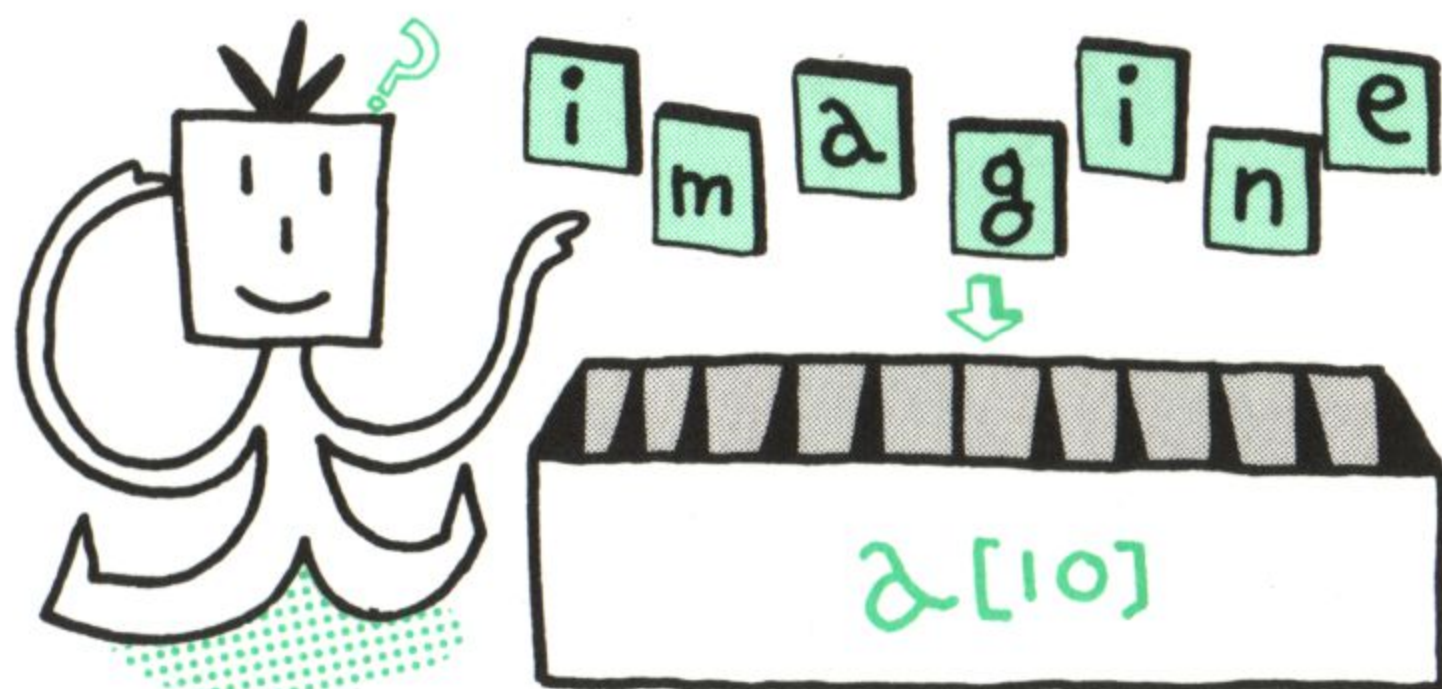
char型変数には、文字１字を入れることができます。これを利用して、長い文章などをデータとして入れたい場合には、**char**型の配列を使います。C言語は、文字列を扱うのに便利な配慮がいろいろとなされています。配列のだいごみは、この**char**型配列にあるといってもよいでしょう。

文章、１つの文節、１つの言葉など、１個以上の文字の集まりを「文字列」といいます。日本語としてはこなれていない言葉ですが、コンピュータ用語としてしばしば出てきますので覚えておいてください。英語でも、文字はcharacter、文字列はstringというように使い分けられています。

配列に文字列「imagine」を入れてみましょう。

◉配列に文字列を入れる――1

```
main()
{
    char a[10];

    a[0] = 'i';
    a[1] = 'm';
    a[2] = 'a';
    a[3] = 'g';
    a[4] = 'i';
    a[5] = 'n';
    a[6] = 'e';

    ……
    ……
}
```

これで７個の文字列が、１個ずつ配列要素に入りますが、１文字ずつ入れるのはめんどうですし、配列を使う意味がありません。配列にまとめて文字を代入するには、次のような方法を用います。

◉配列に文字列を入れる——2

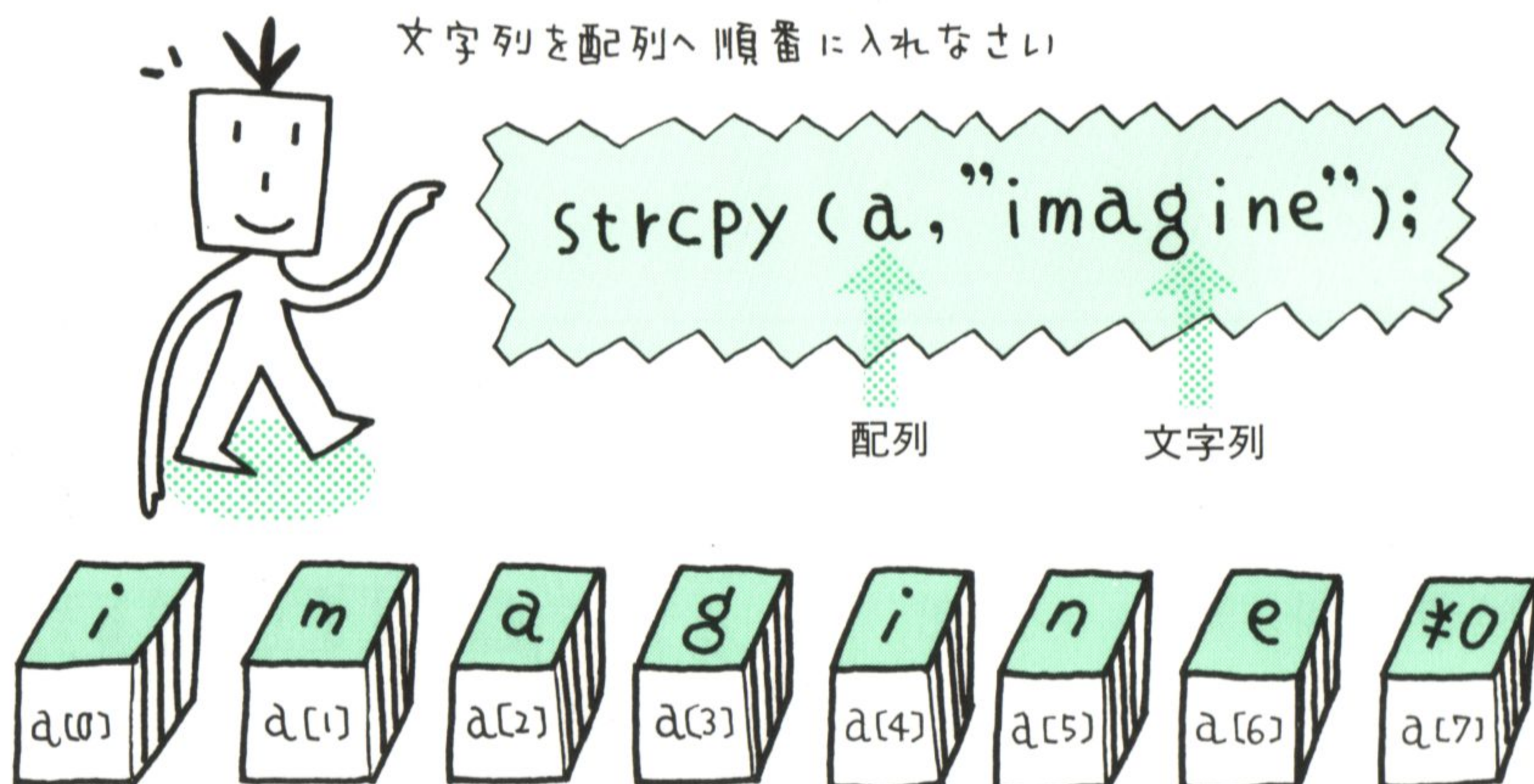
```
#include <stdio.h>   ←標準関数を使うときはこの1行を忘れない

main()
{
    char a[10];   ←配列の宣言

    strcpy(a, "imagine");   ←文字列の代入

    printf("%s", a);   ←文字列の表示
}
```

配列へ文字を代入するとき、

a = "imagine"

などとしたいところですが、これはC言語の文法としては誤りです。**char**(キャラクタ)型変数には、1文字ずつしか代入できないからです。そこで**strcpy**(ストリングコピー)関数というものを使って、

```
strcpy(a, "imagine");
```

と指定すると、配列aの先頭から「imagine」という文字が図のように順に入ります。

図を見ると、入れた覚えのない「¥0」という文字がa[7]に入っています。エラーではないので、訂正する必要はありません。これは、strcpy（ストリングコピー）関数が自動的に入れてくれた記号です。

「¥0」の箱の中身は、実際には「00000000」と、2進数で0が8個入っています。これは、文字列の終わりを示す記号です。文字がいくつかつながっていてその最後に「¥0」があると、そこで文字列が終わりだと判断されます。

もし「¥0」が入っていなくて、a[7]に何かでたらめな値が入っていたとしましょう。画面に配列aを表示するには、プログラムのように「printf("%s", a);」と書きます。しかし、これでaの中身を表示すると、文字列の終わりがわからないため、「imagine…………」と配列aの領域以外のデータまで表示を続け、プログラムが暴走してしまいます。そこでstrcpy（ストリングコピー）関数では、文字列が指定されると自動的に終わりを判断し、「¥0」を入れてくれるわけです。

•••定数

数学と同じように、プログラムにも変数があれば、定数というものもあります。これまで出てきた、

```
a = 'A';
b = 10;
c = 3.14;
```

などの「＝」の右辺は定数です。文字定数の場合は「'」（シングルクォーテーション）ではさむのが決まりでした。文字列定数の場合は、前項で出てきた「"imagine"」のように、「"」（ダブルクォーテーション）ではさみます。

PART3で例にしたサンプルプログラム中の次の2行も、実は文字列定数を表していたのです。

```
printf("整数を２つ入力してください¥n例 1,100");
scanf("%d,%d", &a, &b);
```

C言語のプログラムは、一般にアルファベットの小文字で書かれます。プログラムのステートメントや関数もみな小文字です。変数には大文字を使ってもよいのですが、普通はほとんど使いません。

ところが、ごくまれに大文字が入っているプログラムがあります。これは主に定数を表していて、たいていはプログラムの先頭に次のような一文が書かれてい

るはずです。

```
#define ABC 100 ——①
#define DEF 'Y' ——②
```

①は、「以下のプログラム中でABCという文字は100という数だ」という意味の宣言です。②は「DEFという文字はYという文字だ」という意味です。また右側に式が書かれていることもあります。このようにdefine(デファイン)文は、プログラム中の文字を別の値に置きかえる働きをします。

◉定数を使ったプログラム例

```
/* インチ←→センチを変換する */
#include <stdio.h>
#define INCH 2.54 ←——定数の宣言(INCH=2.54)

main()
{
    int n;
    float x;

    printf("1.ｾﾝﾁ→ｲﾝﾁ    2.ｲﾝﾁ→ｾﾝﾁ  ?");
    scanf("%d", &n);
    printf("x = ");
    scanf("%f", &x);

    if (n == 1)
        printf("%f cm = %f ｲﾝﾁ¥n", x, x / INCH);
    else
        printf("%f ｲﾝﾁ = %f cm¥n", x, x * INCH);
}
```

このプログラムは、1インチ=2.54cmの換算値「2.54」をINCHという定数にして、インチとセンチの変換、表示を行うものです。実行して「1」を選ぶとセンチ→インチ、「2」を選ぶとインチ→センチの変換を行い、結果を表示します。実際に数値を入力して試してみてください。if(イフ)文の中で、定数INCHが変換の計算に使われています。

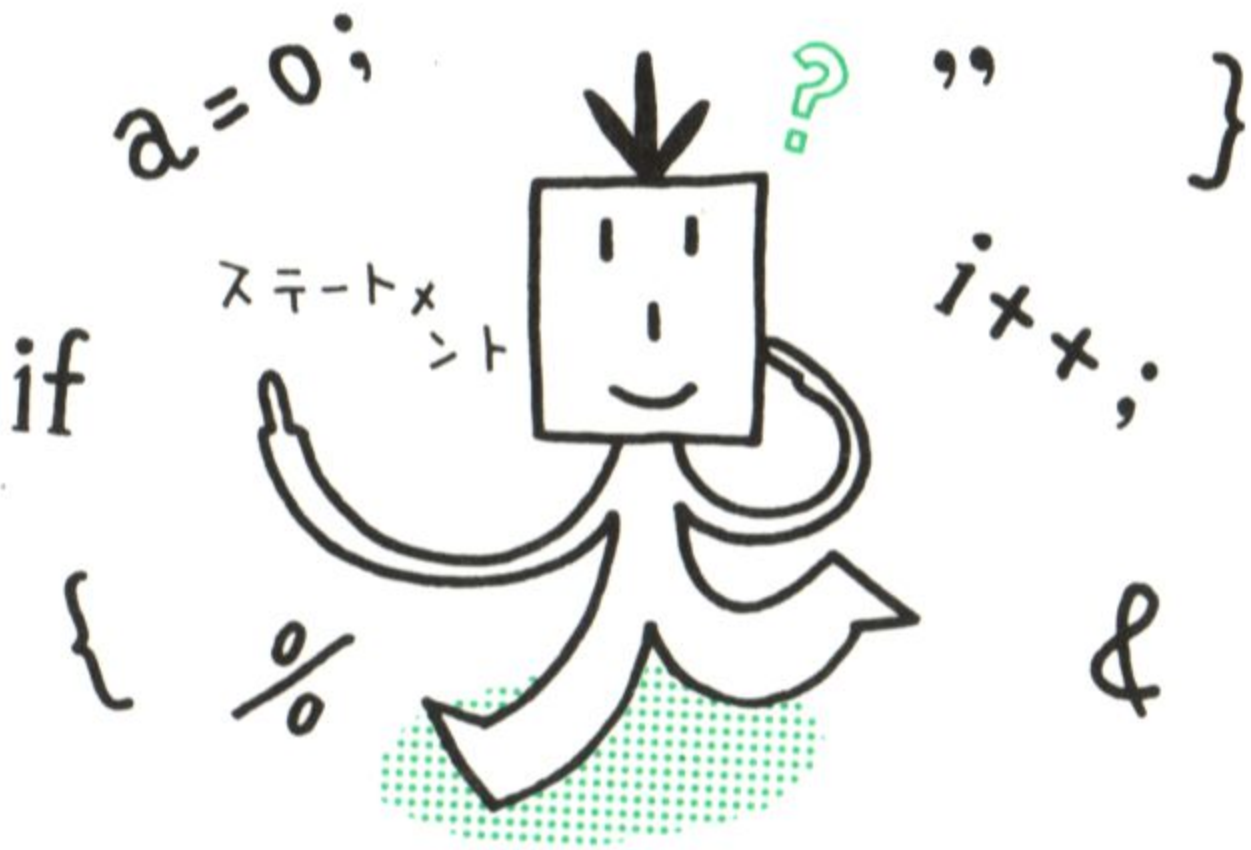
ステートメント

プログラムを作るのに必要な命令語のことを、C 言語では前述したようにステートメントと呼んでいます。ステートメントの数は、BASICの命令語の数に比べてはるかに少ないので、暗記する必要はありません。ステートメントの意味さえしっかり把握しておけば、C 言語のプログラムを解読したり、自分で自由にプログラミングすることができるようになります。

C 言語は、ステートメントから成り立っています。これまでに出てきたステートメントを、いくつかあげてみましょう。「；」までのひとまとまりが、１つのステートメントです。

```
int a, b, c;
printf("整数を２つ入力してください¥n例 1,100");
scanf("%d,%d", &a, &b);
c = 0;
if ( a <= b )
    for( i = a; i <= b; i++ )
        c = c + i;
```

このうち、「printf」「scanf」は、あらかじめ用意された標準関数ですから、純粋なステートメントとはいえません。

以上をまとめると、ステートメントは、プログラムの構成から見た場合、式、複文、空文に分けることができます。

◉ステートメントの分類

式	「a ＝ 0;」「i＋＋;」など演算子を使って表現されたもの
複文	式や他のステートメントが複数集まったもの。「{　}」ではさまれる
空文	式があるはずのところに何もないとき。「；」だけ書かれる

●C言語のステートメント一覧

種　　類	機　　能
if（式）ステートメント	条件に合った場合だけ分岐する
if（式） 　ステートメント else 　ステートメント	条件によって2つに分岐する if～else～を何重にも重ねることができる
while（式） 　ステートメント	式が真のあいだ、ステートメントを繰り返し実行する
do 　ステートメント while（式）	式が真のあいだ、ステートメントを繰り返し実行する ステートメントは最低1度は実行される
for（式1；式2；式3） 　ステートメント	式1～式3の条件のあいだステートメントを繰り返す
switch（式）｛ 　ステートメント case 定数式： 　ステートメント default： 　ステートメント ｝	式の値によって、各case文またはdefault文へ飛ぶ ステートメントの最後に、breakがあればそこで終わり、なければ次の行（case文など）に進む
break；	処理を中断してループやcase文を抜ける
continue；	ループの中で、処理を飛ばして次の回へ進む
return　式；	関数を抜ける。式が書かれているとき、その値が関数の値になる
goto　識別子； 　…… 識別子：文	識別子のある位置へ飛ぶ （gotoはプログラムに使わないほうがよい）

式と演算子

式

ステートメントのうち、いちばん簡単なものが式です。式はどのようなところで、どのような場合に使われるのでしょうか。例として、1から5までの足し算をやってみましょう。

◉足し算のプログラム

```
main()
{
    int c = 0, i = 0;      ← cとiの初期値を0にする

    while(i < 5) {         ← iが5より小さいあいだ｛　｝の中を繰り返す
        i++;               ← iの値を1ずつ増やす
        c = c + i;         ← これが足し算の本体
    }
}
```

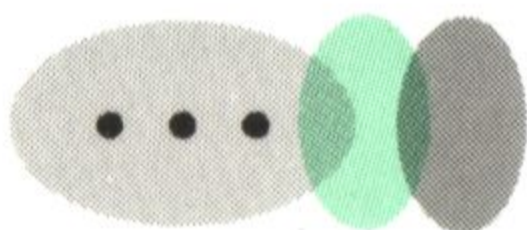
このうち、「c = 0;」「c = c + i;」「i++;」などが式です。「c = 0;」はcに0を代入する、「c = c + i;」はcとiの中身を足してcに入れる、「i++;」はiの値を1増やすという意味です。

iが5より小さいあいだは**while**（ホワイル）ループが繰り返され、iが5になったらループは終わって次へ進みます。実行の過程は、次のとおりです。

1回目のループ	iは1	cは1
2回目のループ	iは2	cは3
⋮	⋮	⋮
5回目のループ	iは5	cは15

ループを抜けたところで、cに入っている数値が答えです。

c = 1 + 2 + 3 + 4 + 5;

と書いても結果は同じですが、足す回数がもっと多くなると、プログラム例の書きかたのほうが短くてすみます。こういう方法はプログラム中でよく使われます。

四則演算——整数型と実数型

次に足し算、引き算、掛け算、割り算などの数値演算を考えてみましょう。

◉数値演算の種類と書式

z = x + y ;	＋	足し算
z = x - y ;	−	引き算
z = x * y ;	×	掛け算
z = x / y ;	÷	割り算
z = x % y ;	%	剰余算（xをyで割った余りを計算する）

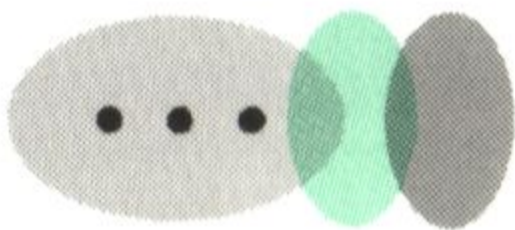
たとえば、先ほどのプログラムを引き算用にしたものが、次のプログラムです。

◉引き算のプログラム

```
main()
{
    int c = 0, i = 0;  ←── cとiの初期値を0にする

    while(i < 5) {
        i++;  ←── iの値を1ずつ増やす
        c = c - i;  ←── これが引き算の本体
    }
}
```

すでにわかるように、答えは−15です。別表にまとめたx、y、zは、プログラムに用いたc、iと同様、すべて整数(**int**)（インテジャ）型で扱ったものです。この場合、割り算をすると、小数点以下は切り捨てられるという約束があります。つまり「3/5」の答えは0で、左辺にある変数には0が代入されます。また剰余算は、変数が整数型の場合だけ計算できる演算で、xをyで割った余りがzに代入されます。

C言語では、変数の型が異なると計算の方法も異なります。もし、x、y、zが実数型の場合、割り算は小数点以下、有効数字の範囲まで計算されますし、剰余算は計算できません。実数とは小数点を含む数で、プログラムでは「float」または「double」で宣言されます。

たとえば、「x = 10」「y = 3」として「z = x / y」を計算すると、答えはx、y、zが整数型の場合「3」、実数型の場合「3.333333」になります。

インクリメントとデクリメント

インクリメントは増加、デクリメントは減少といった意味です。演算のプログラムに出てきた「i＋＋」（または「＋＋i」）が「iのインクリメント」と呼ばれるもので、iの値を1増やします。逆に「i－－」（または「－－i」）は「iのデクリメント」と呼ばれるもので、iの値を1減らします。

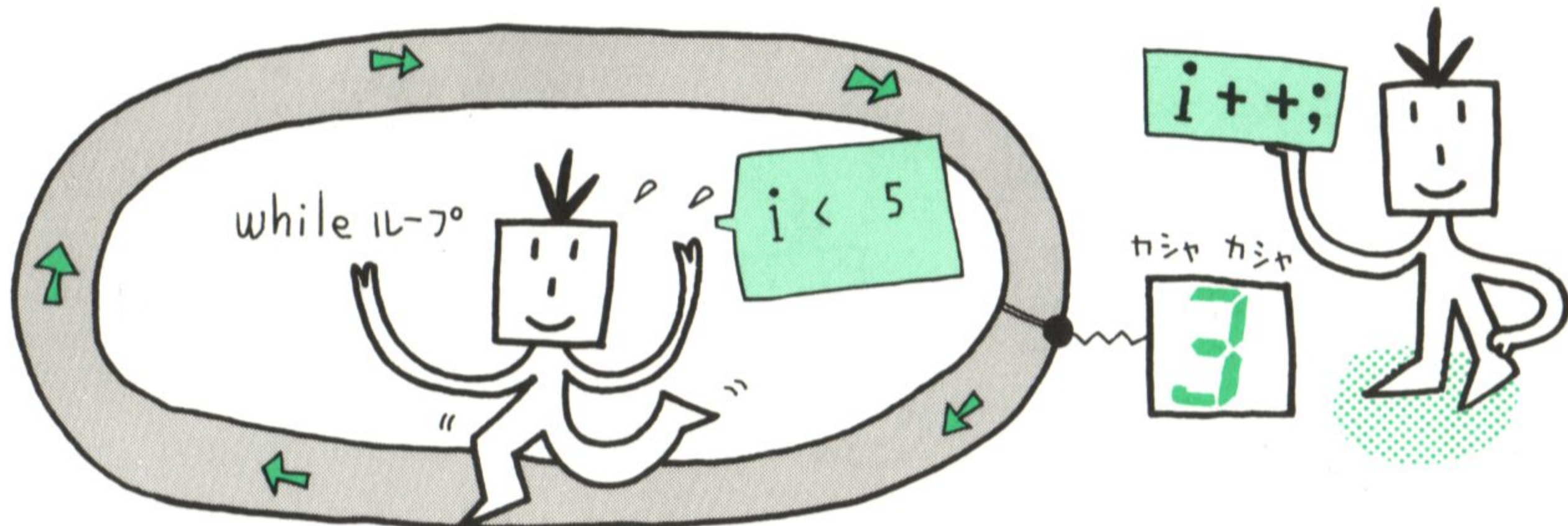
こういった書式は、見慣れないうちはとても違和感がありますし、インクリメントだのデクリメントなど言葉自体もむずかしそうな感じです。しかし、これらはプログラムのいろいろな場面に登場し、強力な演算子として働きます。

◉インクリメントとデクリメントの書式

インクリメント		デクリメント	
i＋＋	＋＋i	i－－	－－i

◉インクリメントを使ったプログラム

```
main()
{
    int c = 0, i = 0;

    while(i < 5) {
        i++;  ←―――― iのインクリメント
        c = c + i;
    }
}
```

このプログラムでは、**while**（ホワイル）ループが1回まわるたびにiの値が1ずつ増えていき、iが5に達するとループから抜けます。

◉インクリメントと式を組み合わせたプログラム

```
#include <stdio.h>

main()
{
    int c = 0, i = 0;

    while(i < 5) {
        c = c + i++;       ←── インクリメント＋式
        printf("i = %d  c = %d¥n", i, c);  ←── 結果の表示
    }
}
```

次に、インクリメント「i＋＋」と式「c ＝ c ＋ i」をくっつけて、働きを調べてみましょう。printf（プリントエフ）文は、実際にプログラムを実行して結果を見るためにつけ加えたものです。これを実行すると、iおよびcの値が以下のように変化します。

i ＝ 1　　c ＝ 0

i ＝ 2　　c ＝ 1

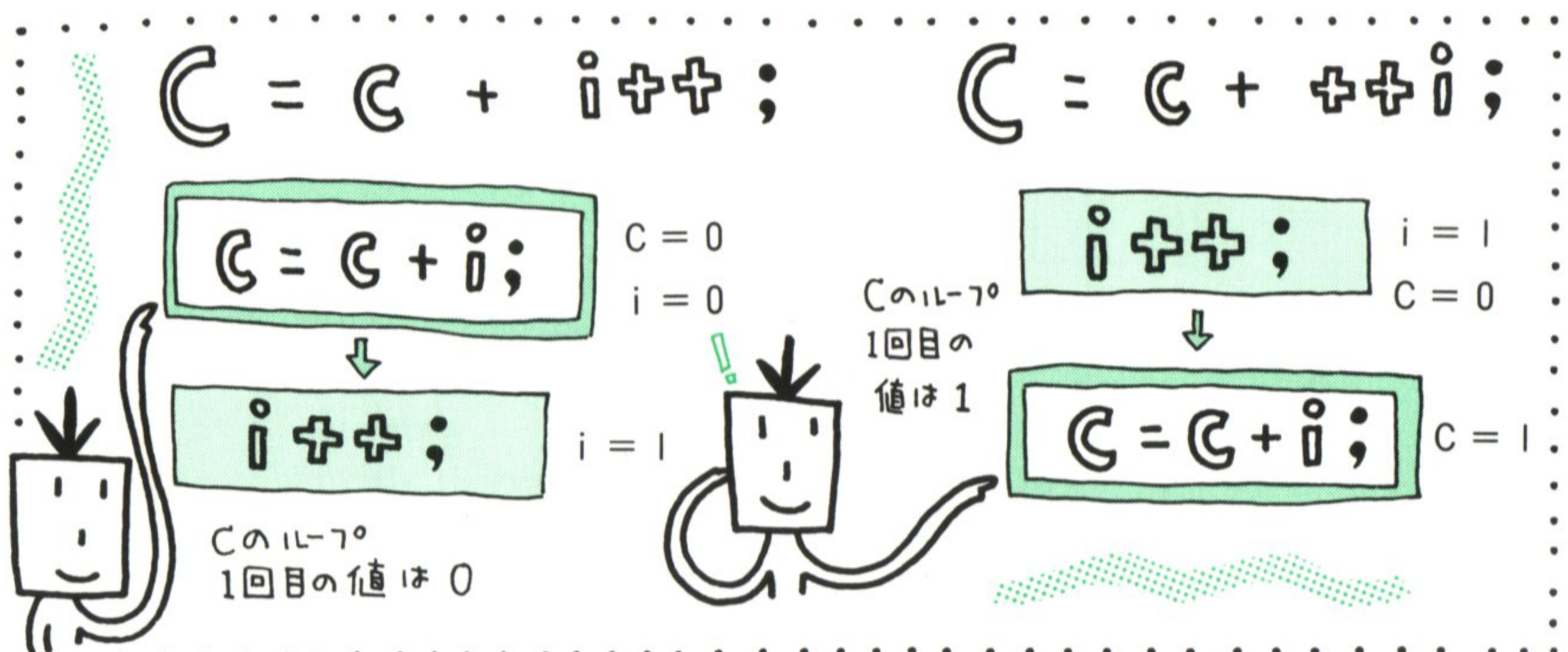
i ＝ 3　　c ＝ 3

i ＝ 4　　c ＝ 6

i ＝ 5　　c ＝ 10

つまり、「c ＝ c ＋ i＋＋;」は、次の順に式が実行されているわけです。

①c ＝ c ＋ i;

②i＋＋;

◉インクリメントの書式を変えたプログラム

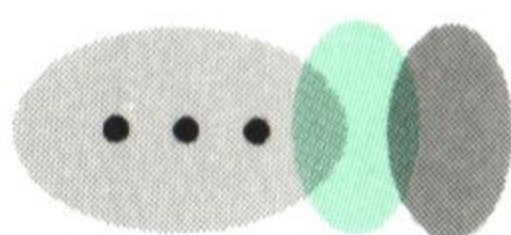
```
#include <stdio.h>

main()
{
    int c = 0, i = 0;

    while(i < 5) {
        c = c + ++i;    ←―――――― ＋記号が前
        printf("i = %d  c = %d¥n", i, c);
    }
}
```

今度は、インクリメントの「＋」記号を前にして、式「c ＝ c ＋i」にくっつけてみましょう。これを実行すると、ｉおよびｃの値が以下のように変化します。

i ＝ 1　　c ＝ 1
i ＝ 2　　c ＝ 3
i ＝ 3　　c ＝ 6
i ＝ 4　　c ＝ 10
i ＝ 5　　c ＝ 15

つまり、「c ＝ c ＋ ＋＋i;」は、次の順に式が実行されているわけです。

①i＋＋;
②c ＝ c ＋ i;

このように、インクリメント演算子を左につけるか右につけるかで、式の実行順序が変わることを覚えておいてください。デクリメント演算子についても、同じです。

・・・大小を比較する

6 比較演算子

数字の大小を比較して処理を変えるような場合、比較演算子が使われます。比較演算子は、たとえば**while**(ホワイル)ループの中などで、これまでにも何度か登場してきました。

たとえば、次のプログラムです。

◉比較演算子を使ったプログラム

```
main()
{
    int c = 0, i = 0;

    while(i < 5) {    ←――iが5より小さいあいだ、{ }の中を繰り返す
        i++;
        c = c + i;
    }
}
```

このプログラムでは、ループを何回まわるかという条件が、比較演算子を使って設定されています。while に続く「(　)」内がそれです。

　i < 5

というのは、ｉが5より小さいという意味です。反対に、

　5 > i

と書くと、5はｉより大きいと読めます。どちらの書きかたでも有効ですが、通常は変数を左に書くことが多いようです。

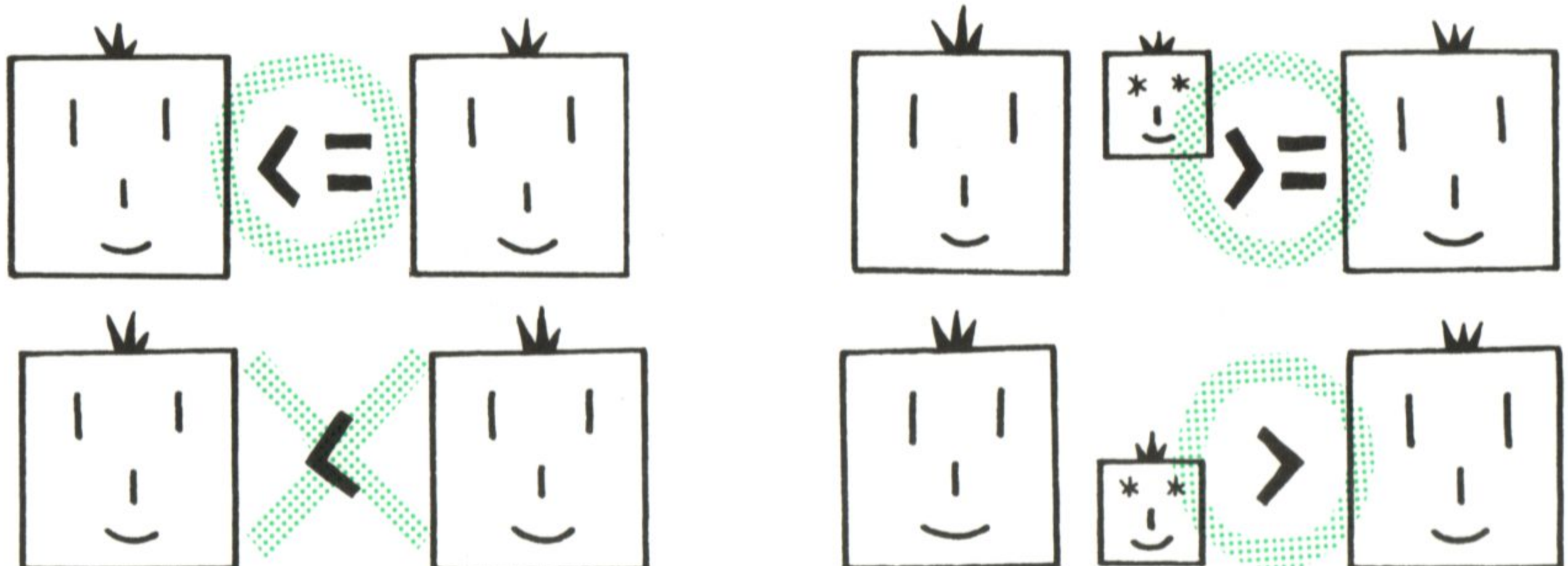
上のプログラム例で、不等号を逆にすると、どうなるでしょうか。

　while (i >5)

こうすると、ｉの初期値が0なので、**while**（ホワイル）ループには1回も入らず、次へ進むことになってしまいます。不等号の向きには十分注意してください。

また、ｉが5より小さいか等しいとき（以下）は、

　i <= 5

と書きます。「<=」は数学の記号では「≦」、「>=」は「≧」と同じ意味です。

6 等価演算子

大小を比較する演算子があれば、等しいかどうかを判定する演算子もあり、等価演算子と呼ばれます。

たとえば、

```
if (i == 0) c = i;
```

という場合、iが0に等しいときは右の式が実行されます。また、

```
if (i != 0) c = i;
```

ならば、iが0に等しくないときだけ右の式が実行されます。

●比較演算子の種類と書式

演算子	意味
<	より小
<=	以下
>	より大
>=	以上

●等価演算子の種類と書式

演算子	意味
==	等しい（等号）
!=	等しくない（不等号）

大小を比較する比較演算子や等価演算子は、例のようにループの条件を判定するカッコの中や、if（イフ）文の条件判定のカッコの中で使われます。

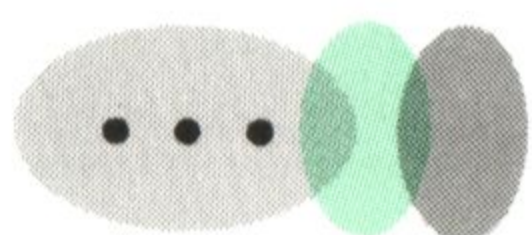
コラム
条件判定はどこで行うか

ループには、すでに触れたように3種類があり、それぞれで条件判定が行われる。

```
①while（条件判定）
        ステートメント
②for（条件判定）
        ステートメント
③do
        ステートメント
  while（条件判定）
```

各ループの性質によって、条件判定と、それによるステートメントの実行が行われるわけだ。

一方、if（イフ）文は以下の3パターンで使われることが多く、やはりそれぞれで条件判定が行われる。

```
①if（条件判定）
        ステートメント
②if（条件判定）
        ステートメント1
  else
        ステートメント2
③if（条件判定）
        ステートメント1
  else if（条件判定）
        ステートメント2
  else if（条件判定）
        ステートメント3
  ……
  ……
  else
        ステートメントn
```

2つ以上の条件設定に使う論理演算子

「iは5以上でかつ10より小さい」という条件は、2つの式で表されます。

```
i >= 5
i < 10
```

この条件を満たしたとき、iに10を代入することを考えてみましょう。つまり四捨五入というわけで、条件が2つあるため、**if** 文を2重にします。

```
if (i >= 5)
    if (i < 10) i = 10;
```

これは、もっと簡単に書くことができます。

```
if (i >= 5 && i < 10) i = 10;
```

このように、いくつかの条件をつなげて書くとき、「&&」など論理演算子と呼ばれるものを使用します。

この例では、不等号の左右を反対にして、

```
( 5 <= i && i < 10 )
```

としてもかまいません。この書きかたのほうがわかりやすいでしょう。「&&」記号は集合のANDと同じ意味で、右の条件と左の条件が同時に成り立つということを示しています。意味からいうと、「かつ、しかも」ということになります。

一方、集合のORを表す記号には「¦¦」が使われます。たとえば、

```
( i < 5 ¦¦ i > 10 )
```

は「iは5より小さいか、または10より大きい」ことを示しています。意味からいうと「または、あるいは」ということになります。

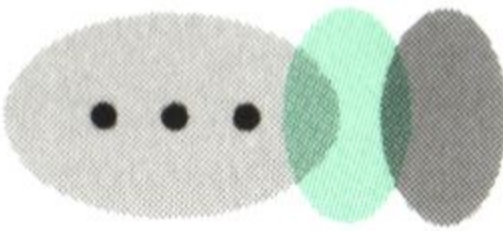
なお、「¦¦」記号は、キーボードの[¦]キーを2回押して入力します。プログラムのなかには、「||」のようにつながった縦棒で表示されているものがありますが、ORを示す場合は、全角の縦棒ではなく、必ず半角の「¦」を使用します。プリンタによっては半角の「¦」をつながった縦棒として印字するものがあるので、そのように表示されてしまうのです。

条件が複数個あるときは、

```
( 5 <= i && i < 10 ¦¦ i == 0 )
```

などのようにつなげて書くこともできます。

●式に使われる演算子一覧

種類と分類		演算子
式		( )　[ ]　->
単項演算子		!　~　++　--　+　-　*　&　sizeof　(型)
二項演算子	乗除	*　/　%
	加減	+　-
	シフト	<<　>>
	比較	<　<=　>　>=
	等価	==　!=
	ビットAND	&
	ビットEXOR	^
	ビットOR	¦
	論理AND	&&
	論理OR	¦¦
条件演算子		?　:
代入演算子		=　+=　-=　*=　/=　%=　>>=　<<=　&=　^=　¦=
カンマ演算子		,

演算子の優先順位

演算子の一覧表は、どういう順序で計算を行うかという演算子の優先順位も示しています。つまり、表の上のほうが優先順位が高く、下にいくほど優先順位が低くなります。また、同じ行で横に並んでいるものは順位が同じで、式の中で出てくる順に計算されます。

6 優先順位の実行例

インクリメントと式をくっつけた例を思いだしてください。

```
c = c + ++i;
```

この場合、iのインクリメントが先で、足し算があとで計算されました。一覧表を見てください。「++」が加減の「+」より上位にランクされています。

単項演算子とは、1つの変数の演算です。ただ、「+」記号は、1やaをわざわざ「+1」「+a」と書くことで、あまり意味がありません。これに対して、「-」

記号は、「－１」「－a」など変数の負の数を表すため、意味があります。

二項演算子とは２つの変数の演算で、たとえば、

a ＋ b

は数学と同じく２つの変数の足し算を表します。

もう少し複雑な式を考えてみましょう。

c ＝ 5 ＋ 20 / 5 ＊ 10－1　　　——①

c ＝ (5 ＋ 20) / 5 ＊ 10－1　　　——②

これらの式で、各演算子の優先順位は次のようになります。

(　)　→ / ＊　→ ＋　－

つまり、一般の数学とほぼ同じだと考えてよいでしょう。①では「/、＊」が優先され、先にある「/」から計算されます。②では、カッコが優先されて足し算が先に行われ、次いで「/、＊」が優先されます。それぞれ①が「c＝44」、②が「c＝49」になることを確かめてください。

次に、論理演算子を含む場合の優先順位を考えてみましょう。

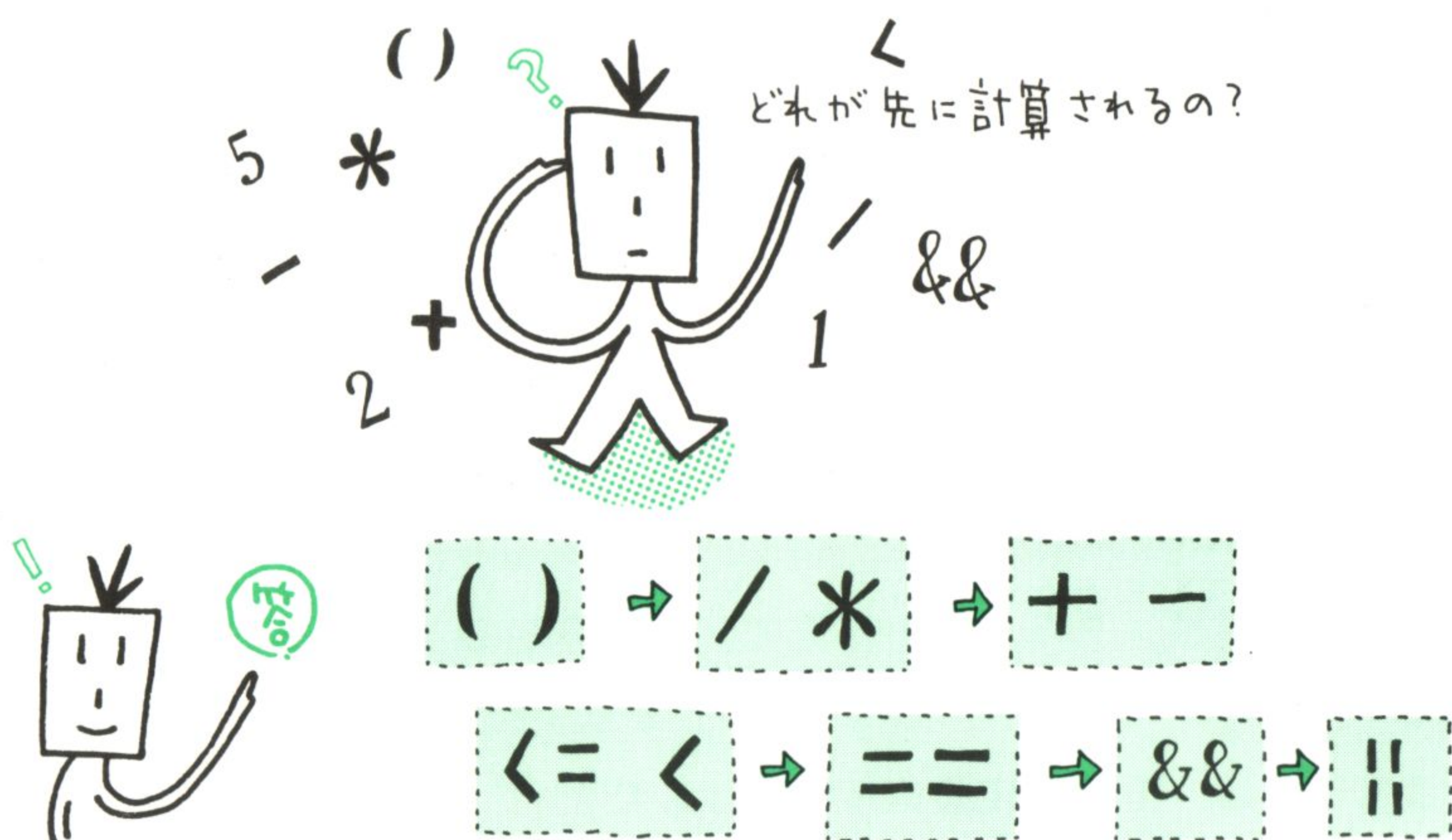
5 ＜＝ i && i ＜ 10 ¦¦ i ＝＝ 0

この式の中の演算子の優先順位は、表を見ると、

＜＝　＜　→ ＝＝ → && → ¦¦

の順です。ですから、

5 ＜＝ i

i ＜ 10

i == 0

の順序で大小を比べたあと、

　「5 <= i」と「i < 10」の「&&」(AND)

　「5 <= i && i < 10」と「i == 0」の「||」(OR)

という順で論理演算が行われます。

6 カッコの使いかた

カッコが最優先されるケースについては、すでに紹介しました。次の式と答えを確認してください。

　c = 5 + 20 / 5 * 100 − 1 = 404

　c = (5 + 20) / 5 * (100 − 1) = 495

また、カッコを2重、3重に重ねると、内側のカッコから先に計算されます。ただし、数学とは違ってカッコに大小の区別はなく、すべて同じカッコ「(　)」を使用します。

　c = (5 + 20 * (4 − 2)) / 5 * (100 − 1) = 891

カッコの優先順位を利用して、式の誤りを防ぐ方法があります。たとえば次のように条件が複数個あり、演算子の優先順位を忘れてしまったような場合、

　5 <= i && i < 10 || i == 0　　　——①

　(5 <= i) && (i < 10) || (i == 0)　——②

②のようにカッコを使って書いておくと、比較が先に行われますから、優先順位をまちがえる心配がなくなります。

分岐

if 文による分岐

if文を使った例は、これまでにもいくつか見てきました。たとえば、

if (i != 0) c = i;

と書かれた場合、iが0でないとき、変数cにiの値を代入するという意味になります。

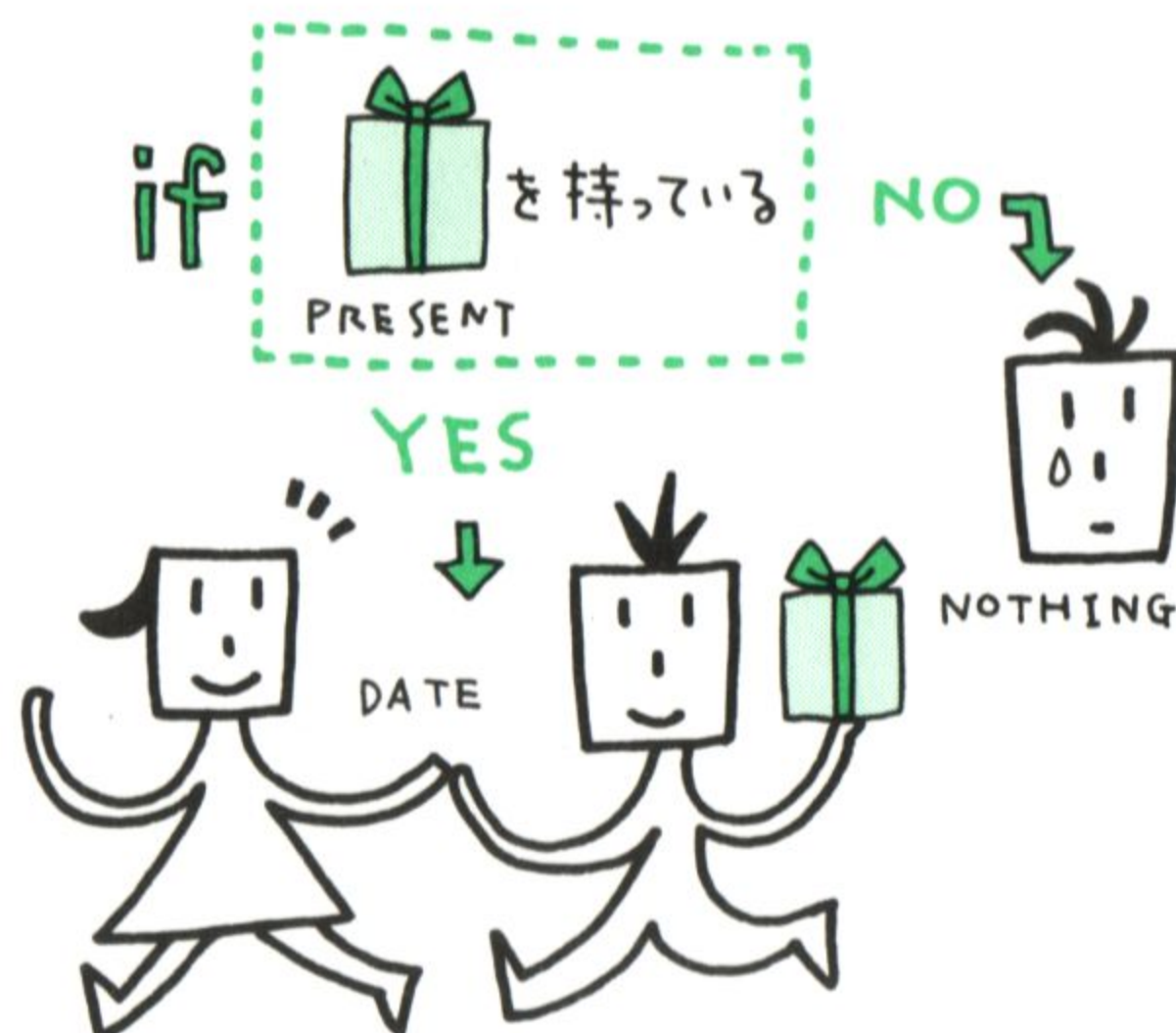
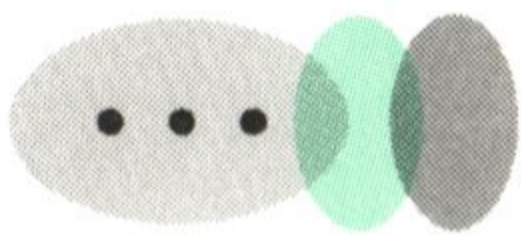
if文は、変数がいろいろに変化するとき、変数の値を判定して分岐する場合に使われます。最も多いのは、ループの中で変数の値を判定するケースです。

◉ if 文で変数を判定して分岐するプログラム

```
main()
{
    int c = 0, i = 0;

    while (i < 100) {
        i++;
        if (i % 5 == 0) c = c + i;
    }
}
```

←剰余で分岐

これは、100までの5の倍数を足すプログラムです。「i % 5」は、iを5で割った余りですから、それが0になるのはiの値が5の倍数のときだけです。このとき、if文はそれに続く式へ分岐して、cにiを足していきます。

whileループでiが99のとき「i++」は100になって最後の足し算を行い、次のループの判定でiは100ですから、そこでループを抜けます。

if~else 文による分岐

左の**if**（イフ）文は、何かが起きたときにちょっと枝別れする、という感じです。この場合、分かれたあと、もとの道へ合流します。

右か左かなど、ふたまたに分かれるときは、**if~else**（イフ・エルス）文を使います。

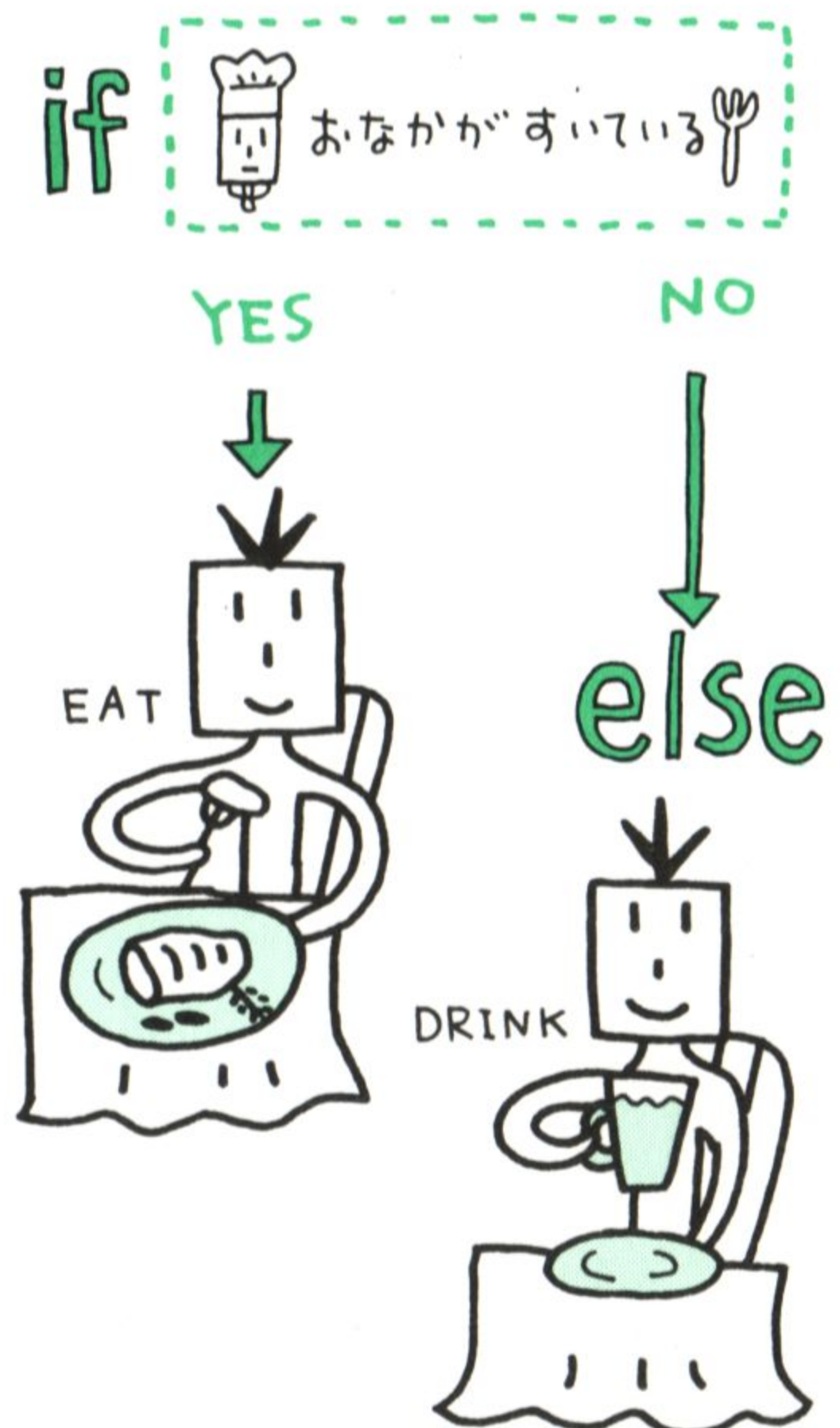
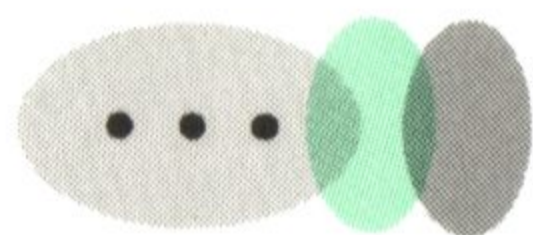
◉ if~else 文による分岐のプログラム

```
main()
{
     int a, b, i;

     a = b = i = 0;
     while (i < 100) {
          i++;
          if (i % 2 == 0) a = a + i;   ←┐
          else b = b + i;              ←┴ 条件で分岐
     }

}
```

このプログラムは、iが偶数のとき（i % 2 == 0）は変数aに、奇数のときは変数bにiの値を足していくもので、aは100以下の偶数の和、bは同じく奇数の和になります。

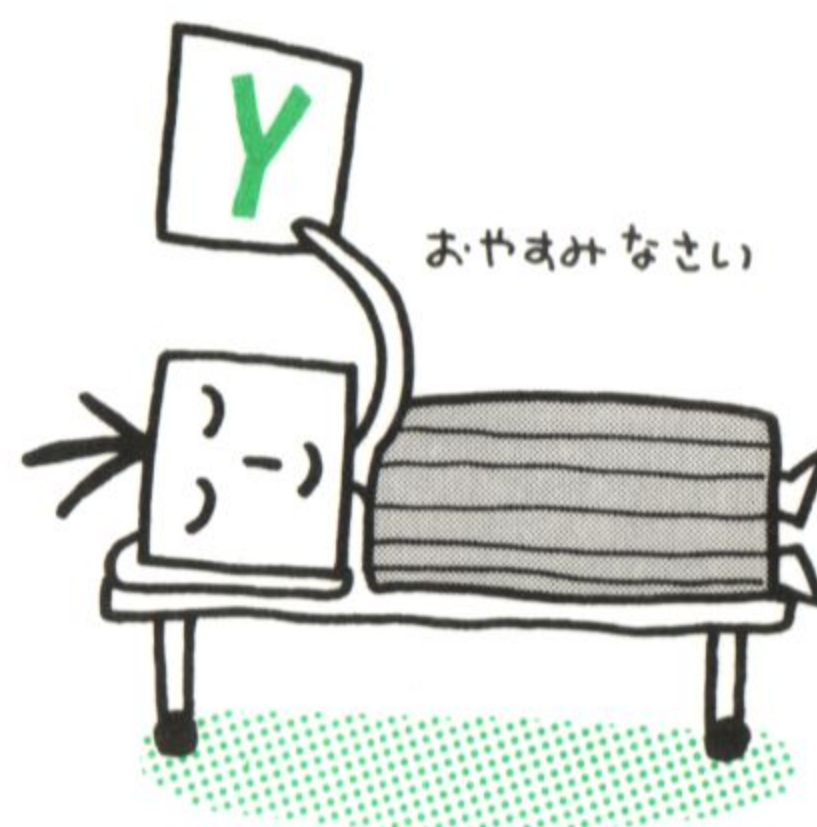
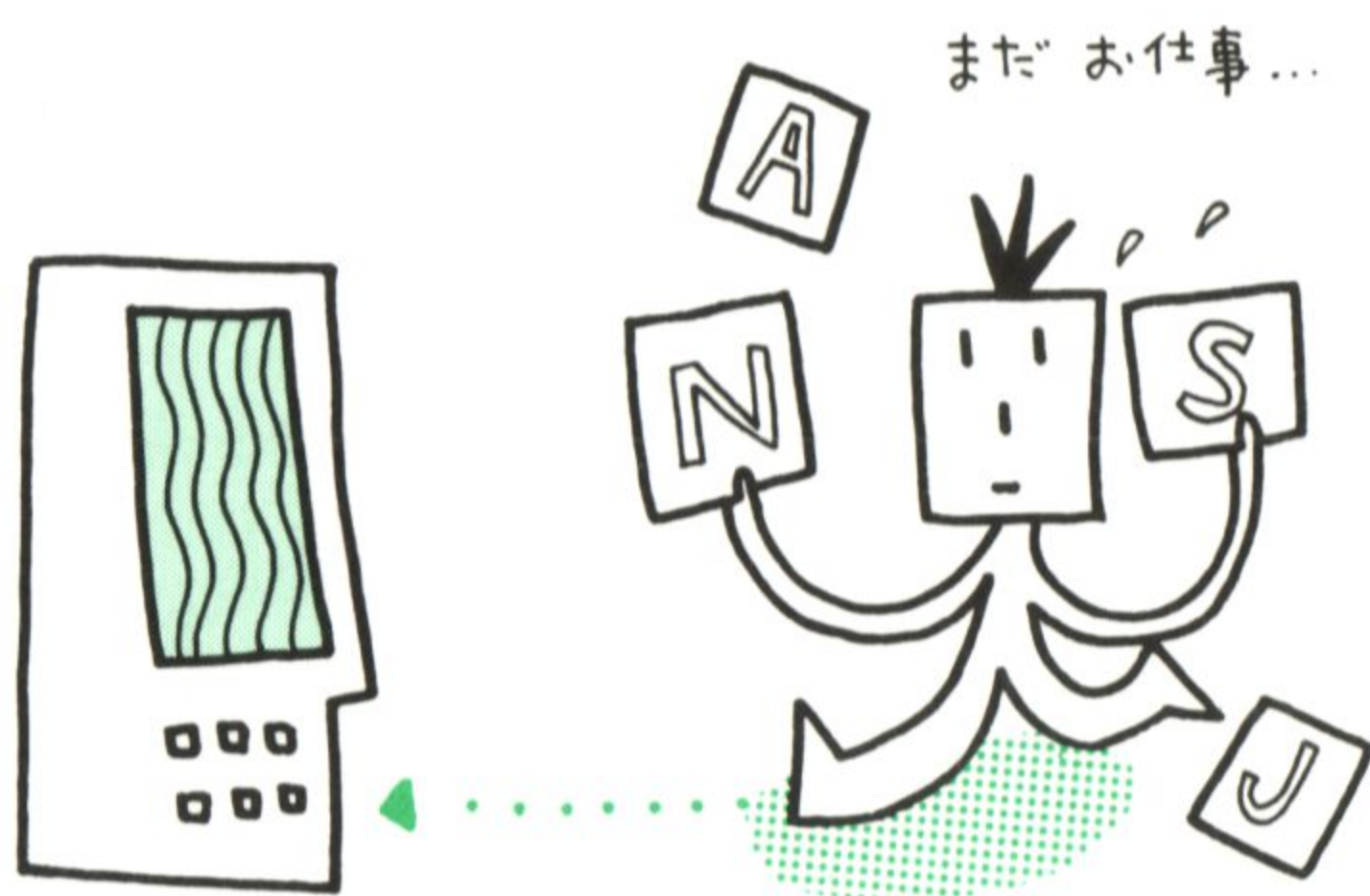
つまり、**if**（イフ）文のカッコ内の条件が正しければ「a = a + i;」が実行され、正しくないときは**else**（エルス）に続く「b = b + i;」が実行されます。

if（イフ）や**else**（エルス）に続く処理が何行にもわたるときは、「{　}」ではさみます。

```
if (条件)    {
       処理 1
       }
else {
       処理 2
       }
```

入力文字を使った分岐

キーボードから入力した文字を判定して分岐させるという方法は、プログラムの中でよく使われます。

◉入力文字による分岐のプログラム例

```
#include <stdio.h>

main()
{
    char x;
    ……
    ……

    printf("終了しますか？<y/n>");  ← メッセージの表示
    x = getchar();                   ← 入力された文字をxにセット

    if (x == 'y' || x == 'Y') {      ┐
        …… 終了処理 ……               │
    }                                │
    else  {                          ├ 分岐処理
        …… 継続処理 ……               │
    }                                │
}                                    ┘
```

押されたキーを判定し、それによって異なる処理に分岐するプログラムです。最初の**printf**(プリントエフ)文で、次の表示を出します。

```
終了しますか？<y/n>
```

「x = getchar( );」というステートメントの**getchar**(ゲットキャラクタ)関数は、キーボードから入力された文字を変数（この場合はx）にセットする働きをします。セットされた文字は**if**(イフ)文で判定され、xの値が小文字のyまたは大文字のYのときは次に進み、終了処理を行います。xがyまたはY以外のときは**else**(エルス)文に分岐し、継続処理を行います。

プログラムの終了を確認させるときに、よく使われる処理です。

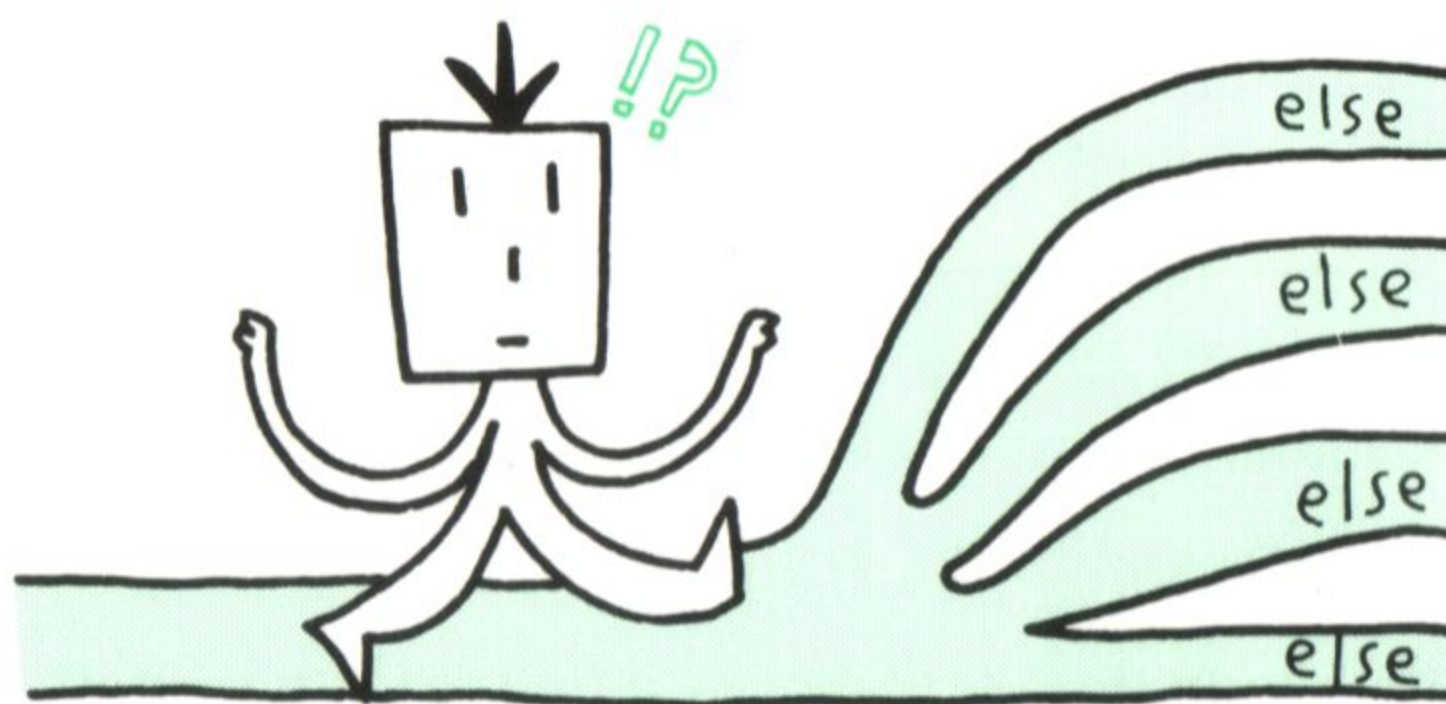
if〜else文が何重にも続く場合の分岐

分かれ道がいくつもある場合は、**if〜else**(イフ・エルス)文を何重にも使って分岐させることができます。

◉ if〜else文を何重にも使った分岐のプログラム

```
#include <stdio.h>

main()
{
    int i = 0;

    while (i < 5){
        i++;

        if (i == 1) printf("a");
        else if (i == 2) printf("b");
        else if (i == 3) printf("c");
        else if (i == 4) printf("d");
        else  printf("e");
    }
}
```

iが0から5まで変わるあいだ、「a〜e」の文字を画面に表示するプログラムです。iの初期値は0なので、ループに入ると「i++」によって「i=1」となり、

if (i == 1) printf("a");
が成立して「a」が表示されます。2回目以降はiが1ずつプラスされ、それぞれの値に対応したelse文によって「b～e」の文字が表示されていきます。

switch～case文による分岐

前例のように、if～else文による分岐が何重にも繰り返されるとき、代わりにswitch～case文を使うこともできます。

◉ switch～case文による分岐のプログラム

```
#include <stdio.h>

main()
{
    int i = 0;

    while (i < 5){
        i++;

        switch (i) {
            case 1: printf("a");break;
            case 2: printf("b");break;
            case 3: printf("c");break;
            case 4: printf("d");break;
            default: printf("e");
        }
    }
}
```

switch～case文は、switch文に続く変数の値によって、それに対応するcase文を実行します。この場合、switch文の変数にiが使われています。iが1のときはcase 1へ、3のときはcase 3へというように飛び先が変わります。そして、それぞれのprintf文を実行して、画面に「a～e」の文字を表示していきます。

各行末の「break」はなんでしょう？　それには、次のようにプログラムを変えてテストをしてみます。

◉「break」を省略したswitch～case文のプログラム

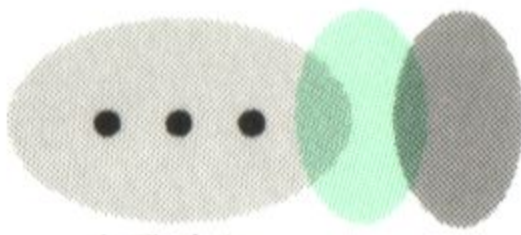
```
#include <stdio.h>

main()
{
    int i = 0;

    while (i < 5){
        i++;

        switch (i) {
            case 1: printf("a");
            case 2: printf("b");
            case 3: printf("c");
            case 4: printf("d");
            default: printf("e");
        }
    }
}
```

これを実行すると、最終的に次のように表示されます。

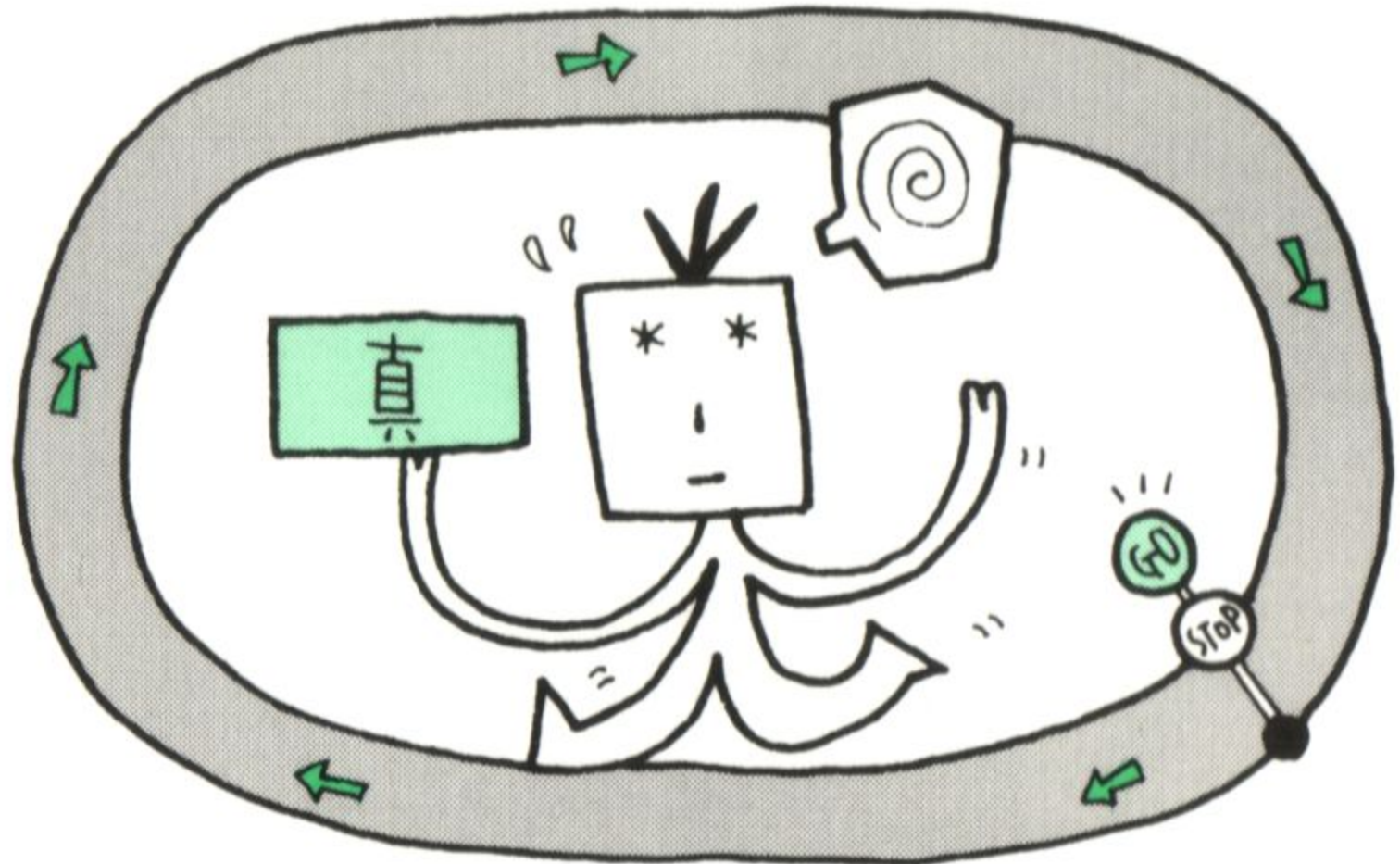
```
abcdebcdecdedee
```

break（ブレーク）文には、**switch**（スイッチ）文のブロックを抜ける働きがあります。また、**switch**（スイッチ）**～case**（ケース）文は**break**（ブレーク）文がないと、次の**case**（ケース）文を続けて実行します。この例がそれで、1回目のループではcase 1から順に「abcde」、2回目はcase 2から「bcde」、3回目はcase 3から「cde」というように、飛び先の**case**（ケース）文以降をそのつど最後まで実行しているわけです。**switch～case**（スイッチ・ケース）文は、この点で**if～else**（イフ・エルス）文とは異なりますから注意してください。

最後の**default**（デフォルト）文は、変数の値が**case**（ケース）文で示す値以外のときに実行されます。なお、**switch**（スイッチ）文に続く変数（i）や**case**（ケース）文に使われる定数は、例の場合、整数値でしたが、ここには文字を指定することもできます。

ループ

while ループ

while(ホワイル)ループは、これまでにもたくさん出てきました。C言語のループの代表は、このwhile(ホワイル)ループです。while(ホワイル)ループの書式は、次のようです。

```
while (式)
    ステートメント
```

式には通常、ループをまわる条件が書かれます。ループの最初に条件判定を行って、条件が正しいあいだ(真)、ループは何度もまわり続けます。条件が正しくなくなれば（偽)、そこでループを抜けて次の行へ進みます。

ステートメントが2つ以上あるときは、「{　}」ではさみます。

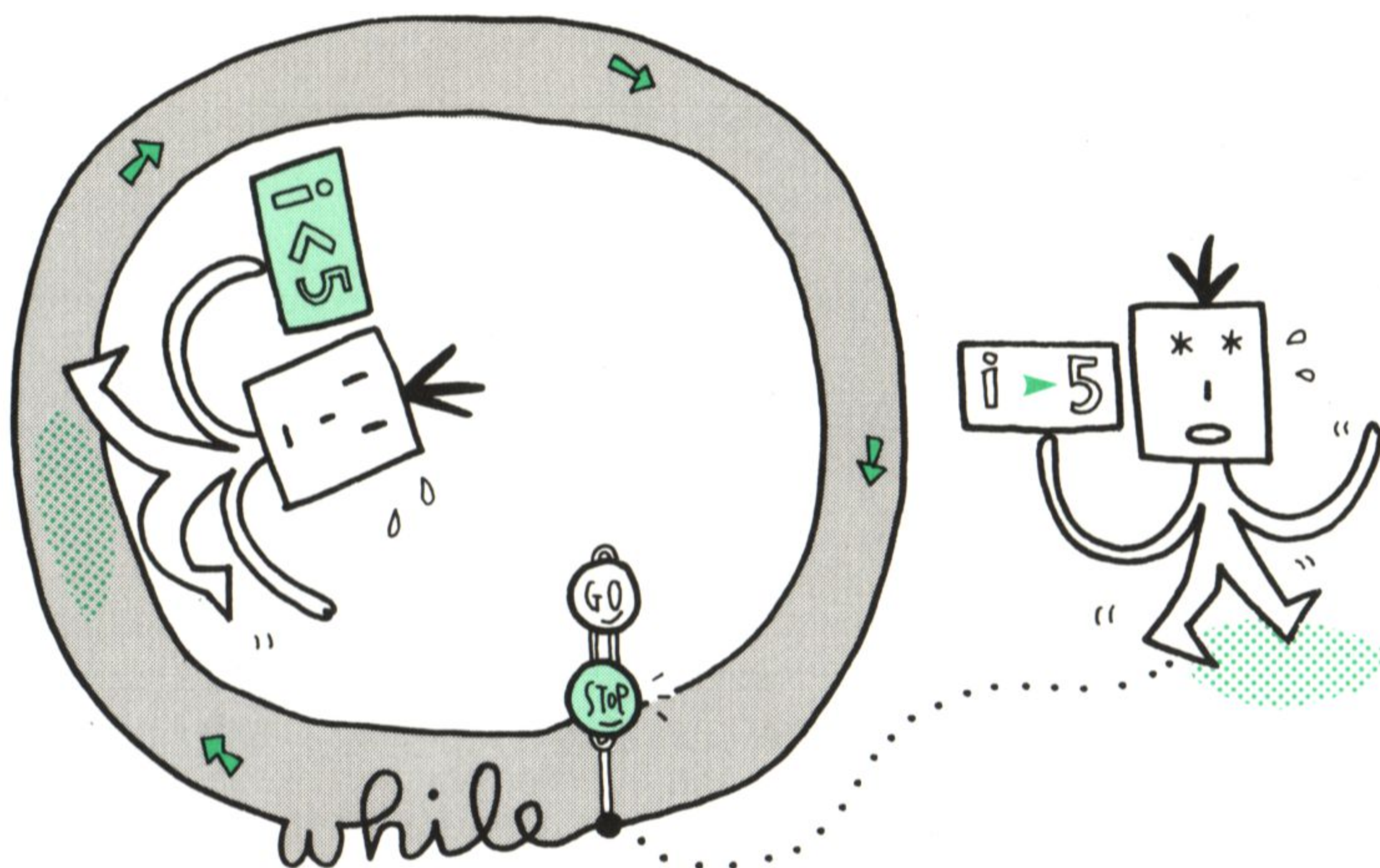
次のループは、iが5以下の場合にまわり、iが5になると「i＜5」の条件に合わないのでループを抜けます。while(ホワイル)ループは、ループをまわる回数をあらかじめ決められない場合に使われます。

◉ while ループのプログラム

```
main()
{
    int c = 0, i = 0;

    while(i < 5) {
        i++;
        c = c + i;
    }

}
```

ステートメントが2つあるとき｛　｝ではさむ

◉式をまとめた while ループのプログラム

```
main()
{
    int c = 0, i = 0;

    while(i < 5)
        c = c + ++i;  ←―― 式とインクリメント

}
```

for ループ

for（フォー）ループは、ループをまわる回数があらかじめわかっているときに使います。

◉ for ループのプログラム

```
main()
{
    int c = 0;

    for (i = 0; i <= 100; i += 5)
        c = c + i;
}
```

すでに説明した、5の倍数を足し合わせる計算を、**for** ループで書いたプログラムです。**for** 文に続く、代入演算子「+=」を使った、

　　i += 5

は、

　　i = i + 5

と同じ意味です。つまり、iの値は5ずつ増えていきます。そこで、

　　for (i = 0; i <= 100; i += 5)

は、iの値が0から始まって5ずつ増えていき、iが100になるまでループがまわることを示します。つまり、iの値は次のように変化するわけです。

　　i = 0

　　i = 5

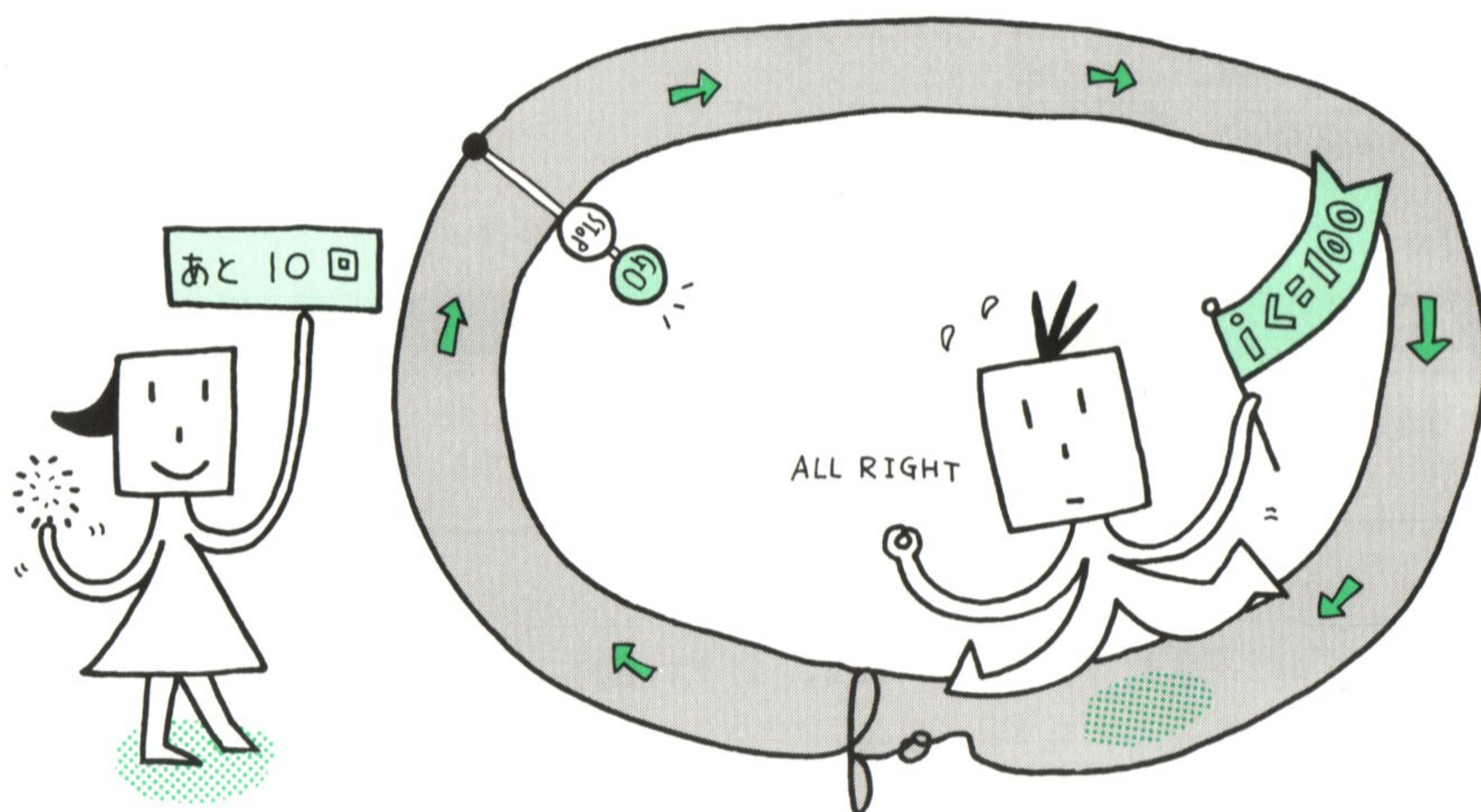
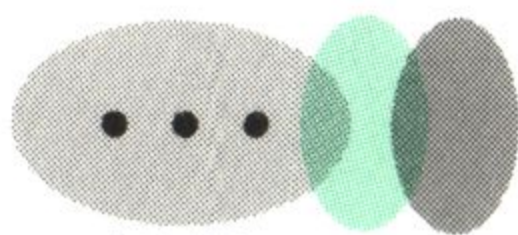
　　i = 10

　　i = 15

　　　⋮

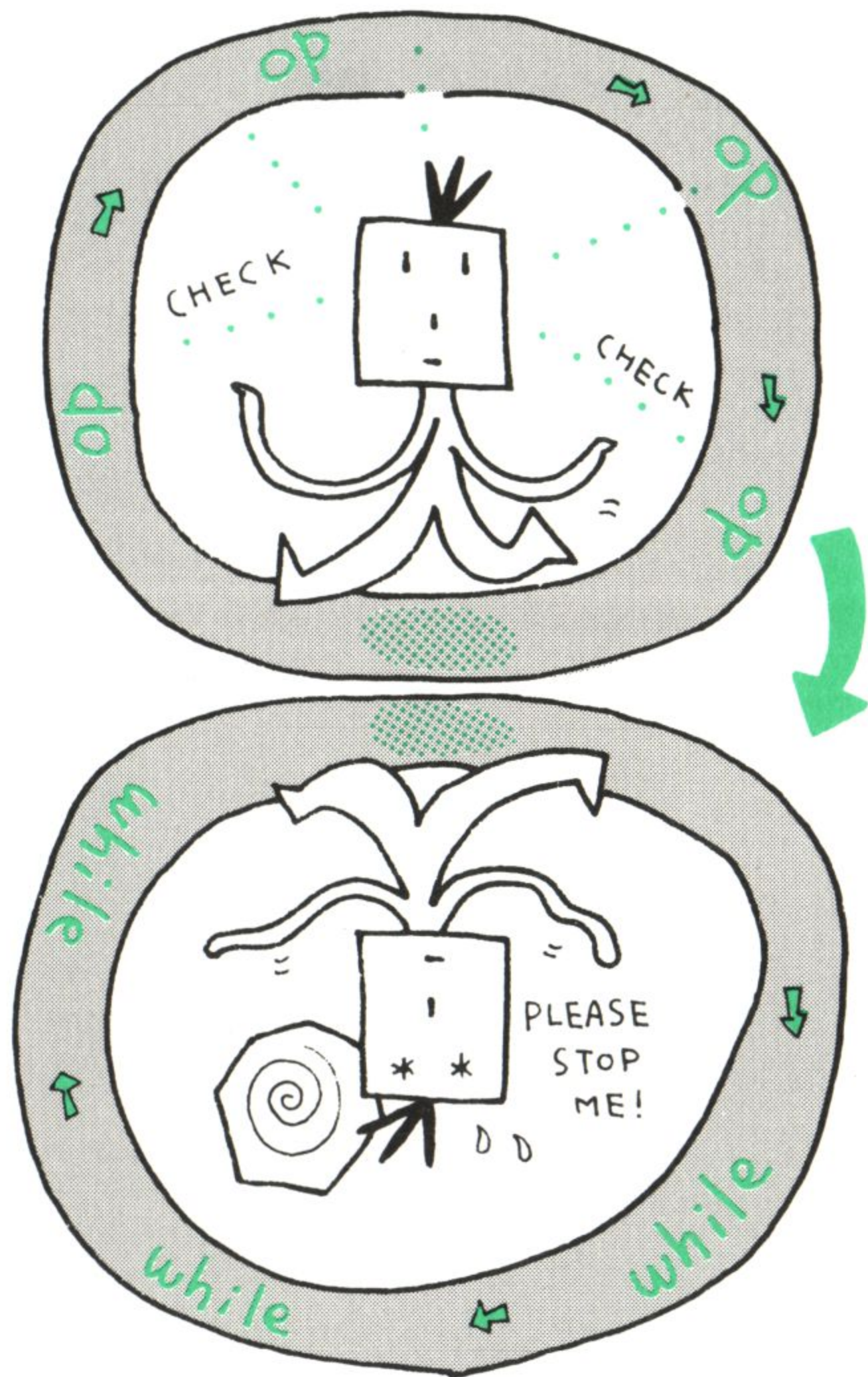
for ループの書式は、次のようです。

```
for（初期値；判定条件；増分）
    ステートメント
```

for 文に続くカッコ内には、ループがまわるあいだ、変化する変数についての条件を書きます。

do〜while ループ

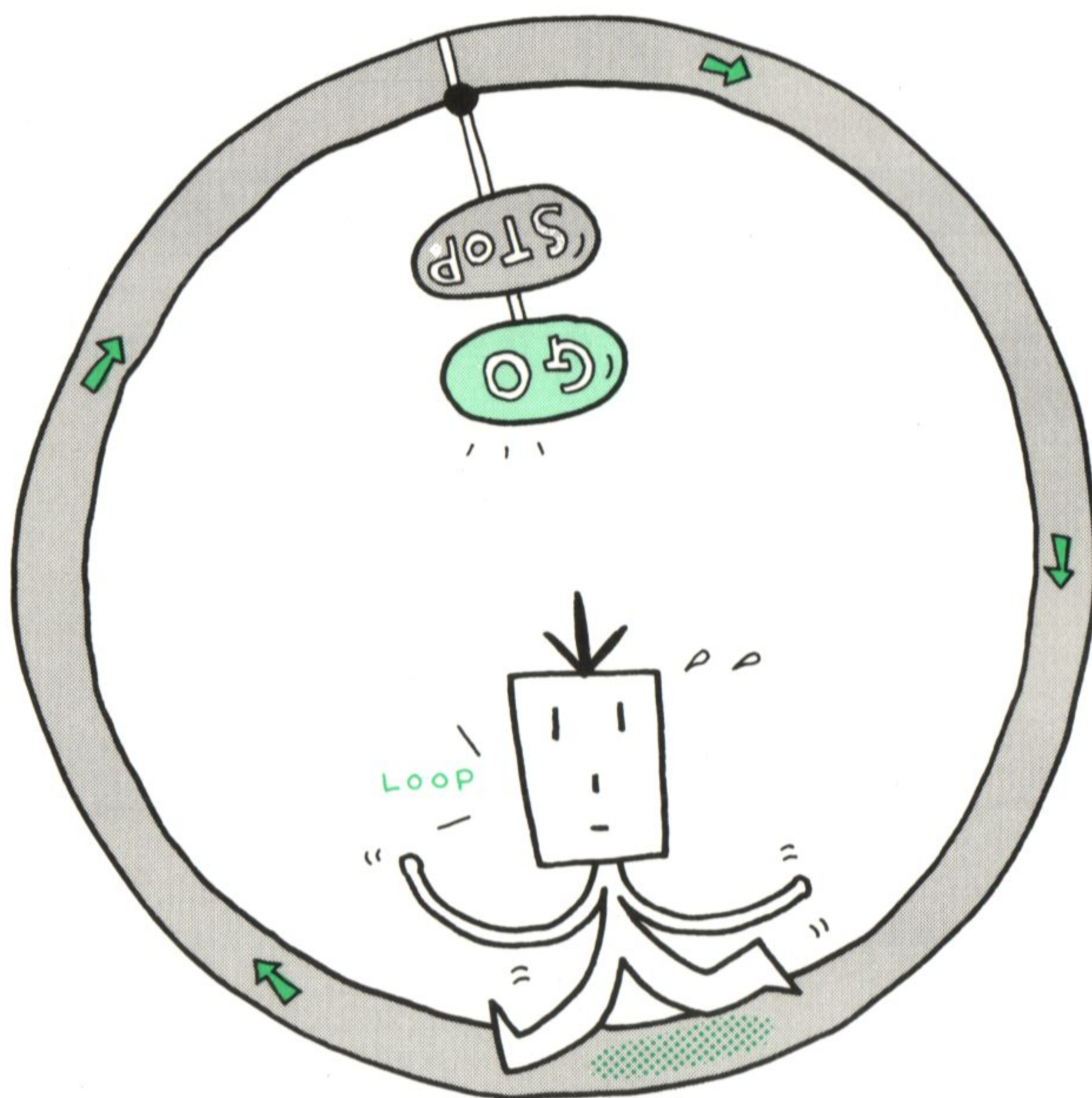
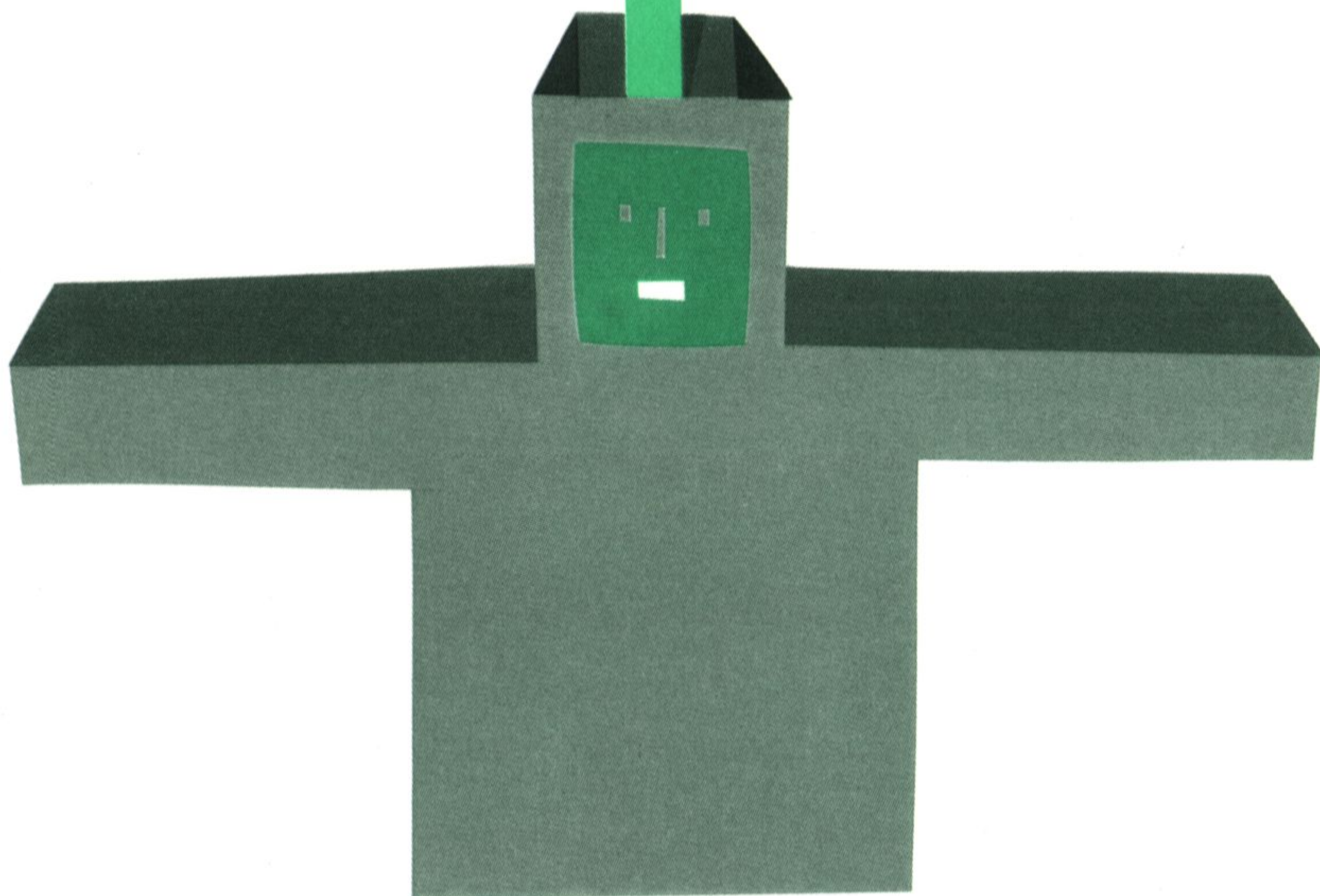
while（ホワイル）ループは最初に条件判定を行うため、条件によってはループを１度も通らないことがあります。これに対して、必ず１回はループを実行するのが**do〜while**（ドゥ・ホワイル）ループです。書式は、次のようです。

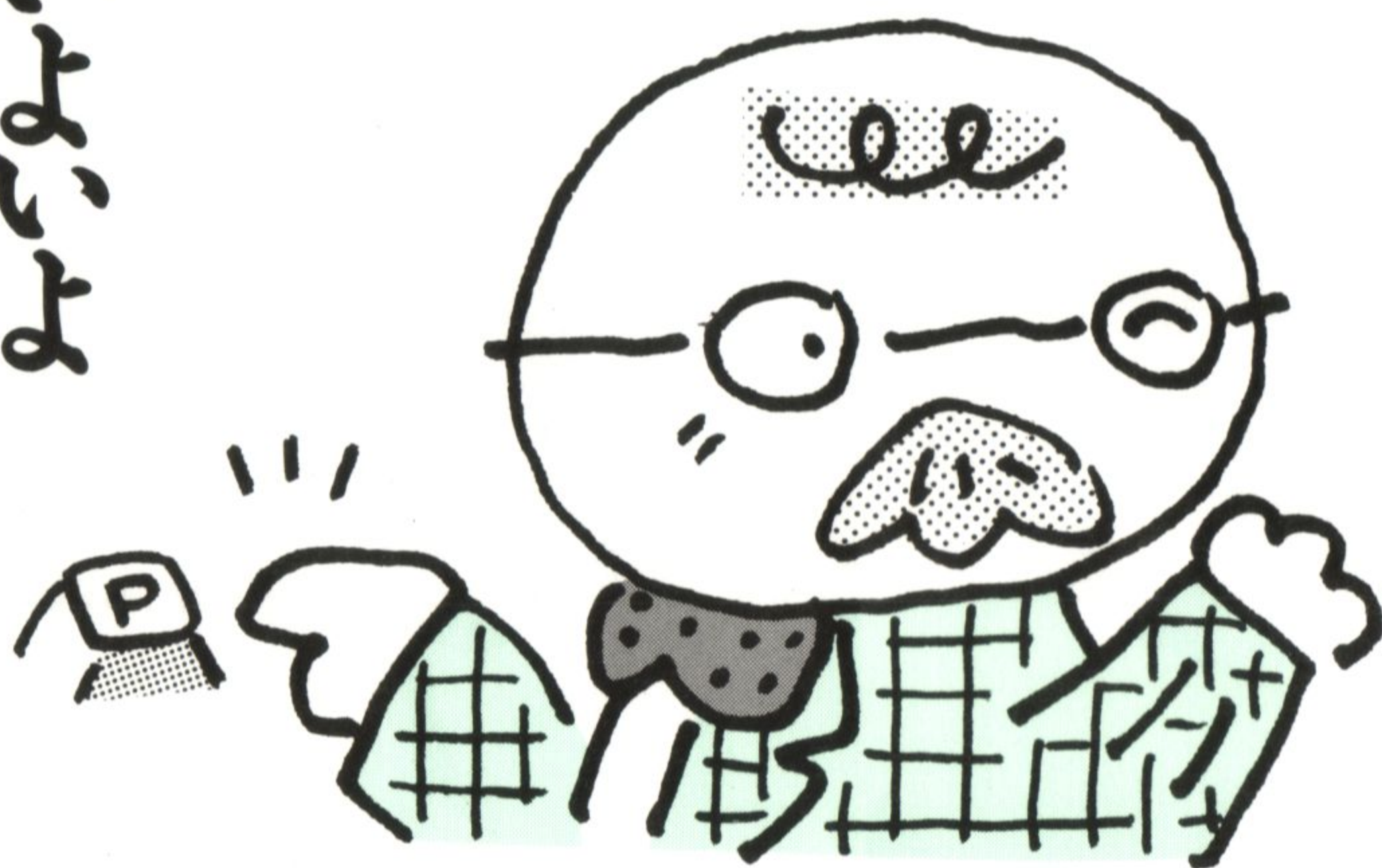
```
do
      ステートメント
while（条件判定）
```

最初に１回、ステートメントを実行し、そのあと条件の判定を行って、条件に合っていれば、再び同じステートメントを実行します。これを何度も繰り返すのが、**do〜while**（ドゥ・ホワイル）ループです。

◉ループを何度も繰り返す do〜while ループのプログラム

```
/* 指定のキーを押すまで何度もループを繰り返す */
#include <stdio.h>

main()
{
    char x;

    do {
        printf ( "何かキーを押してください" );
        x = getchar();
    } while ( x != 'Q' );

}
```

このプログラムでは、メッセージの表示のあと、キーボードから入力された文字が、113ページで説明した**getchar**（ゲットキャラクタ）関数によって変数xにセットされます。次の**while**（ホワイル）文でxが「Q」かどうかが判定され、「Q」の場合はループを終了します。つまり、キーボードから「Q⏎」が入力されるまでループがまわり続け、メッセージを表示して入力を待つという処理を繰り返すわけです。

PART 5

プログラミングのテクニック……2

いよいよプログラミングのポイントへ。

玲子　PART4では、C言語の基本的なプログラミングのテクニックを説明したわね。ここに出てきたのは、ほかのどの言語にも共通に備わっているものなのよ。だから、これらの機能を知っておけば、ほかの言語もだいたいわかるというわけ。

田川　へー、それじゃ**if**(イフ)文や**while**(ホワイル)文なども同じ書きかたをするのかい？

玲子　書きかたは言語によっていろいろ違うのよ。ちょうど、「私」という言葉が英語ではI、フランス語ではJe、ドイツ語ではIchと違うようにね。でも、プログラミング言語は基本的な構文の似たものが多いわね。BASIC(ベーシック)とC言語の対比は前に説明したから、もう一度見直すといいわね（24ページ参照）。

田川　そうか。前に見たときは何のことだかチンプンカンプンだったけれど、意味がわかってくると、なるほどという感じだね。

玲子　そこで、もう少し突っ込んで、C言語特有の性質——関数、ポインタ、構造体について説明していくわね。

田川　なかなかむずかしそうだね。しかし、関数なら**printf**(プリントエフ)とか**scanf**(スキャンエフ)とか……すでにいろいろ出てきたぞ。

玲子　そう。ただし、**printf**(プリントエフ)や**scanf**(スキャンエフ)は“標準関数”と呼ばれているものだけど、このようにあらかじめ用意されているものを使うばかりじゃなく、関数は自分で作ることもできると説明したはずよ。関数とは機械の部品のようなもので、既製品を使うこともあれば、自分で作る部品もあるし、人の作った部品を改良して使うことさえあるのよ。

田川　関数には少し親しみを感じてきたけど、自分で作るのはむずかしそうだね。

玲子　基本的なことさえ知っていれば簡単。たとえば「足し算をする関数」とか「引き算をする関数」なんかだと、すぐにできるのよ。こういうのは、いかにも簡単そうでしょ。

田川　…………。

玲子　まず初めに**printf**（プリントエフ）や**scanf**（スキャンエフ）など代表的な標準関数の使いかたを整理して、次に関数の作りかたを説明することにしましょう。それから、ポインタと構造体よ。ポインタはC言語特有のもので、「これがあるからCがアセンブラと同じように使える」という、なかなかすばらしい機能なのよ。

田川　おいおい、ポインタというのは名前からしてむずかしそうだし、これは人に聞いたんだが、ポインタの使用例は理解しにくく、ポインタがいちばんの難関だというじゃないか。われわれ初心者が、そこまでやる必要があるのかい？

玲子　ポインタがむずかしいのは確かよ。でも、ポインタを知っているかどうかが、C言語をわかるかどうかの分かれ目なのよ。なるべく簡単な例で説明するから、ぜひポインタの考えかたを理解してほしいの。

田川　ま、それはそれとしても、次の構造体というのが、またいかめしい名前だなぁ。

玲子　名前はそうでも、いま話題のデータベースを理解する糸口になる、といえば興味がわくんじゃないかな。構造体についても、基本的な部分だけをやさしく説明するわね。

標準入出力関数の使いかた

▼書式つき出力関数

printf｜プリントエフ

printf 関数は、すでに何度も出てきました。「文字を画面に出す働きをする」ということは、漠然とでも理解できていることでしょう。

printf 文の書きかたにはいろいろあり、右のカッコ内の書式が少しずつ違っています。

```
printf("a");                                          ──①
printf("整数を2つ入力してください¥n例 1,100");          ──②
printf("%dから%dまでの和を計算します", a, b);           ──③
printf("答え  : %d", c);                              ──④
```

上記の①～④について、画面への表示結果とそれぞれの働きを、順に解説していきましょう。

❶ printf("a")；──「"」記号ではさむ

表示結果→

```
a
```

カッコ内の、「"」(ダブルクォーテーション)ではさまれた文字を、画面に表示します。

なお、これと同じ働きをする、

```
printf("hello,world¥n");
```

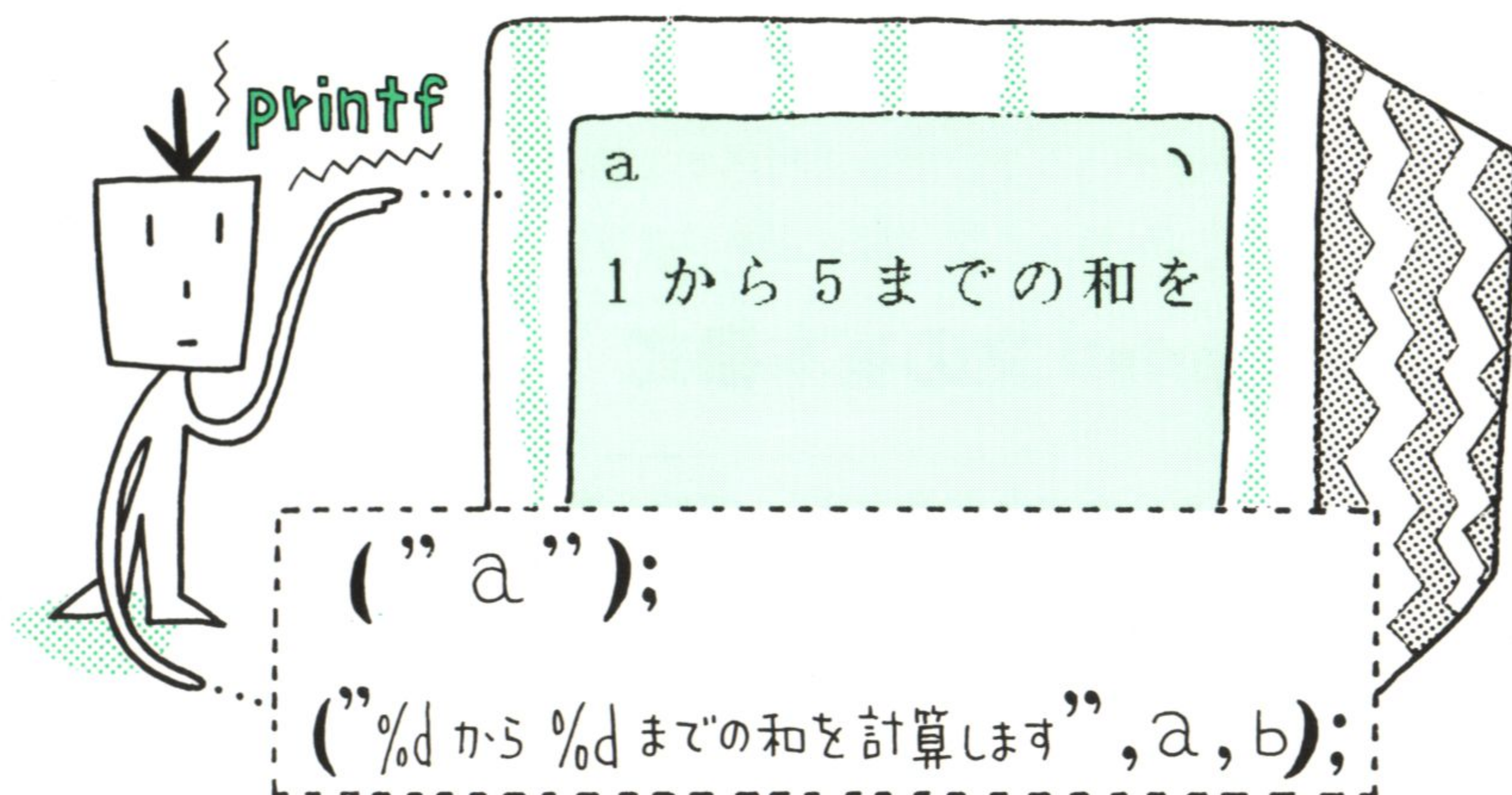
は、次のように文字を画面に表示します。これは、C言語の決定版マニュアルといわれるカーニハン&リッチーの著作『プログラミング言語C』に登場する有名な一文です。**printf** 文の末尾に、「¥n」という気になる文字がありますが、これについては、②で説明します。

```
hello, world
```

printf

a 、

1から5までの和を

("a");

("%dから%dまでの和を計算します", a, b);

❷ printf("整数を2つ入力してください¥n 例　1,100"); ——改行指定「¥n」

表示結果→
```
整数を2つ入力してください
例　1,100
```

①とは異なり、表示が2行に分かれています。これが「¥n」の働きで、「"」ではさまれた文字の中に「¥n」があると、その位置で改行を行います。このように「¥」のついた文字を制御文字といい、「¥n」はこれで1つの文字を表しています(実際には16進数で0cという1バイトのコード)。

❸ printf("%dから%dまでの和を計算します", a, b); ——変換文字「%d」

「"　"」の文の次に、カンマで区切られた変数aとbが並んでいます。そして、「"　"」の中には、これに対応するかのように「%d」が2つあります。もしaとbの変数が1と5の値を持つとき、表示は次のようになります。

表示結果→
```
1から5までの和を計算します
```

変数aとbは実際に2つの「%d」に順に対応しており、初めの「%d」にはaの値が、2番目の「%d」にはbの値が入って表示されます。このような書きかたが、**printf** 文の標準的なものです。

次の④も、これと同じ結果になります。

❹ printf("答え　:%d", c); ——変換文字「%d」

表示結果→
```
答え　:15
```

この「%d」のように「%」で始まる文字を変換文字と呼びます（書式指定子と呼ばれることもある）。「%d」は、対応する変数を10進数で表示する働きをします。変換文字はプログラム中に用いて対応する値に置きかえられるもので、後述するように、いくつかのタイプ（型）があります。

printf 文の基本書式

printf("表示したい文字と変換文字"，変数，変数，……)；

「"　"」内に記述する変換文字の数は、あとに並べた変数の数と一致していなければなりません。この数が一致するとき、変換文字の位置に変数の値が置きかえられます。「"　"」内にはまた、「¥n」や「¥t」などの制御文字が入ることもあります。

変数を用いないで使うこともありますが、そのときは変換文字も省略します。

変換文字の種類と働き

変換文字は **printf** 文など書式つき入出力関数に用いられるもので、「%d」のほか、いろいろな種類があります。それというのも、変数には**char**（キャラクタ）型、**int**（インテジャ）型、**float**（フロート）型、**double**（ダブル）型という区別があるうえ、それぞれを配列で用いることもあり、いろいろなタイプの変数があるからです（85ページ参照）。それに対応して、変換文字にもいろいろな種類があります。プログラム例を見ながら、変換文字の種類と働きを紹介していきましょう。

◎ char 型変数を入れるプログラム

```
#include <stdio.h>
main()
{
    char a;

    a = 'A';        ←――char 型変数に文字を代入するときは「'　'」ではさむ

    printf("変数aの中身は%cです", a);
}                              ↑
                               └―― 変換文字で、変数 a に対応している
```

↓実行結果

変数aの中身はAです

6 変換文字「%c」──1文字に対応

変数aは、char(キャラクタ)(文字)型です。aに文字Aが代入されますから、これをそのまま表示するために「%c」という変換文字を用います。

次に、いろいろなタイプの変数を並べて表示してみましょう。

◉いろいろな変換文字を用いたプログラム

```
#include <stdio.h>
main()
{
    char a, e[6];          ←── 文字型の変数宣言
    int b, f, g;           ←── 整数型の変数宣言
    float c;               ←── 4バイト浮動小数点数型の変数宣言
    double d;              ←── 8バイト浮動小数点数型の変数宣言

    a = 'A';
    b = 5;
    c = 1.23;
    d = 4.567e-8;          ←── 4.567×10⁻⁸のこと
    e[0] = 'H';
    e[1] = 'e';
    e[2] = 'l';
    e[3] = 'l';
    e[4] = 'o';
    e[5] = '¥0';
    f = 021;               ←── 8進数の定数。先頭に「0」をつける
    g = 0x4a62;            ←── 16進数の定数。先頭に「0x」をつける

    printf("変数aの中身は%cです¥n", a);
    printf("変数bの中身は%dです¥n", b);
    printf("変数cの中身は%5.3fです¥n", c);
    printf("変数dの中身は%5.3eです¥n", d);
    printf("変数eの中身は%sです¥n", e);
    printf("変数fの中身は%oです¥n", f);
    printf("変数gの中身は%xです¥n", g);
}                                ↑
                  変数は対応する変換文字の指示によって表示される
```

↓実行結果

```
変数aの中身はAです
変数bの中身は5です
変数cの中身は1.230です
変数dの中身は4.567e-008です
変数eの中身はHelloです
変数fの中身は21です
変数gの中身は4a62です
```

この例は、いろいろな変数の型と、それを表示する変換文字を表しています。変数aは**char**(文字)型、b、f、gは**int**(整数)型、cは**float**（4バイト浮動小数点数)型、dは**double**（8バイト浮動小数点数)型の変数です。

また、変数eは**char**型の配列として宣言されています。

6 変換文字「%f」「%e」――実数、指数形式に対応

cとdに対応する変換文字は、

%5.3f

%5.3e

と書かれています。この「%」と「f」や「e」の間にある数字は、表示する桁数の指定です。この場合の「5.3」とは、全部で5桁のうち小数点以下は3桁までを表示するという意味で、次のような形になります。

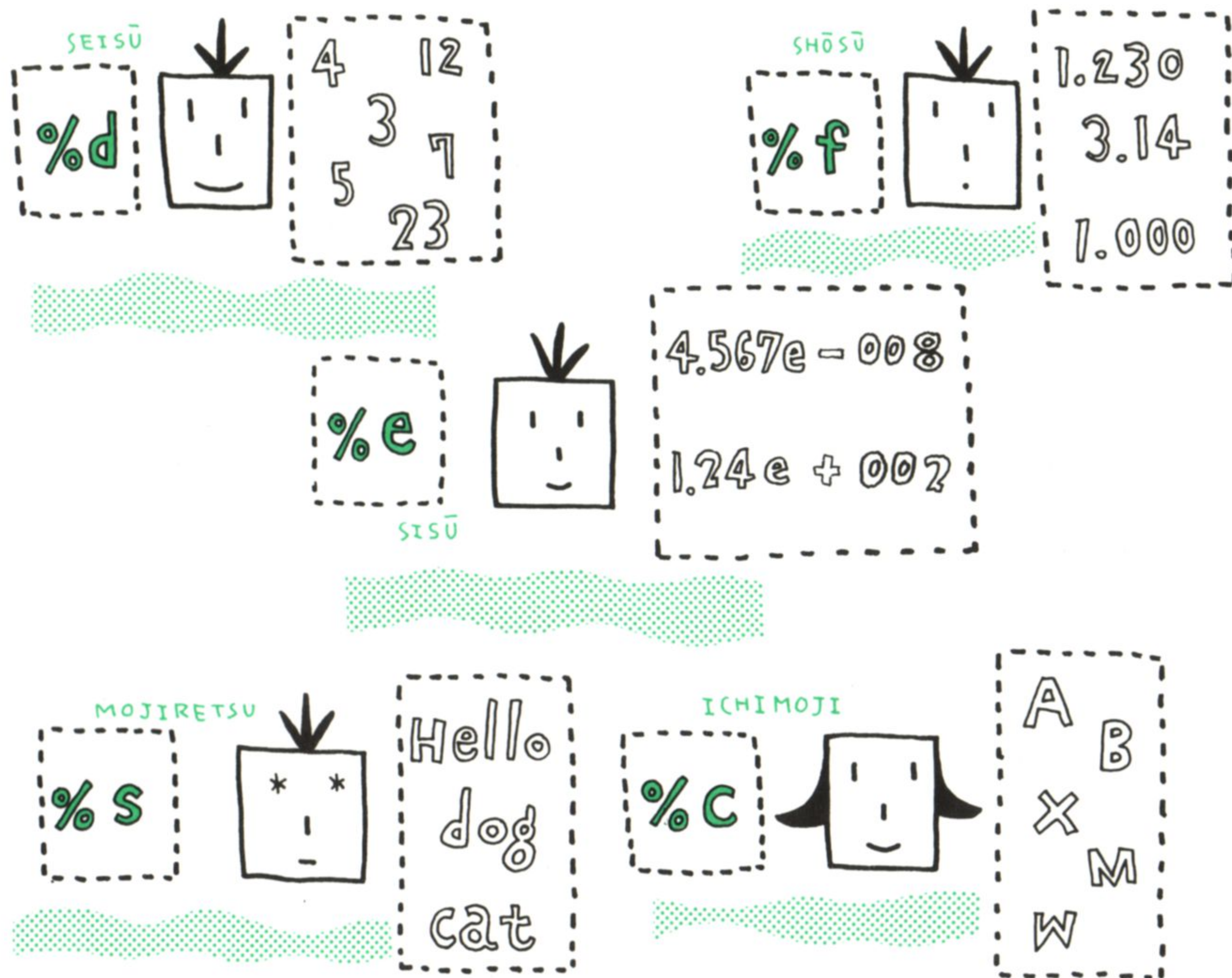
```
        小数点以下3桁
             ↓
       ○.○○○
        ↑
  ピリオドを入れて全部で5桁
```

ですから、「%5.3f」は変数cの数値を、

1.230

というように表示したわけです。

変数dに対応する「4.567e-8」という指数の書きかたはほかのプログラミング言語でも使われ、数値「4.567×10^{-8}」を表します。これを「%5.3e」と指定して表示すると、5桁のうち小数点以下は3桁までを表示するという意味なので、

4.567e-008

と表示します。

では、変換文字に桁数を指定する数字をつけないで、「%f」「%e」のままで使うとどうなるでしょう。

```
printf("変数cの中身は%fです¥n", c);
printf("変数dの中身は%eです¥n", d);
```

↓実行結果

```
変数cの中身は1.230000です
変数dの中身は4.567000e-008です
```

桁数を指定しない場合は、あらかじめ決められた形式で表示されますから、このように書いてもエラーではありません。小数点以下何桁まで表示されるかは使用するコンパイラによって異なり、たとえば上記のようになります。

「%f」（数値を実数形式で表す）、「%e」（数値を指数形式で表す）といった変換文字は、画面にどう表示するかという指定だけを行うもので、**float**（フロート）型が「%f」、

double(ダブル)型が「%e」を使うということではありません。ですから、次のように、変換文字を逆に指定してもよいのです。

```
printf("変数cの中身は%5.3eです¥n", c);
printf("変数dの中身は%5.3fです¥n", d);
```

↓実行結果

```
変数cの中身は1.230e+00です
変数dの中身は0.000です
```

変数dの値は、実数に変換すると0.00000004567なので、小数2桁以下が四捨五入され、表示が「0.000」になったわけです。

6 変換文字「%s」——文字列に対応

次の「%s」は、配列eに対応しています。配列eの文字列が順に代入されて、中身が「Hello」と表示されたわけです。配列に文字を入れる方法は、ほかにもいくつかあります（ふろく：188ページ参照）。ここでは、文字を1つひとつ代入しました。

e[5]に代入した「'¥0'」は、数字の0（16進数では00と表現する）を代入するための記号です。これは実際の2進数ビットパターンでは、

00000000

となります。

同じ変数の内容を、1文字対応の「%c」で表示してみましょう。

```
printf("変数eの中身は%c%c%c%c%cです¥n",
      e[0], e[1], e[2], e[3], e[4]);
```

↓実行結果

```
変数eの中身はHelloです
```

結果は同じですが、「%s」を使うほうが簡単です。「%s」は、配列に代入された文字列を、「¥0」(16進数00）の前まで表示する変換文字です。

もし文字列の最後に「¥0」がない場合は、配列の個数を超えて文字表示を続けることになり、プログラムは暴走してしまいます。配列に文字を代入するときは、最後の1文字ぶんに必ず「¥0」を入れておかなければなりません。

◎主な変換文字、制御文字一覧

変換文字

変換文字は、書式つき入出力関数で使用される。対応する変数と変換文字の型が合っていないとき、printf は変換文字の型に合わせて変数を表示し、scanf はまちがった値を入力する

変換文字	対応する変数	printf での使用例
%c	char 型 1 文字	a ='A'; printf ("変数 a の中身は%c です", a) ; →変数 a の中身は A です
%s	char 型文字列	strcpy(s, "Hello") ; printf("変数 s の中身は%s です", s) ; →変数 s の中身は Hello です
%d	int 型　10進数	b = 5 ; printf("変数 b の中身は% d です", b) ; →変数 b の中身は 5 です
%f	float 型、double 型 ○○.○○（実数）	c =1.23 ; printf("変数 c の中身は%5.3 f です", c) ; →変数 c の中身は1.230です
%e	float 型、double 型 指数部をつけて表示	d=4.567e-8 printf("変数 d の中身は%5.3e です", d) ; →変数 d の中身は 4.567e-008です
%o	int 型　8 進数	f=021; printf("変数 f の中身は% o です", f) ; →変数 f の中身は21です
%x	int 型　16進数	g=0x4c printf("変数 g の中身は% x です", g) ; →変数 g の中身は 4c です

制御文字

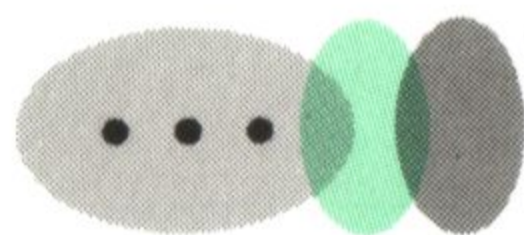
制御文字	名　前	使用例
¥n	改行コード	printf("終わり¥n さようなら") ; →終わり 　さようなら
¥t	タブコード	printf("終わり¥t さようなら") ; →終わり　　　さようなら

▼書式つき標準入力関数

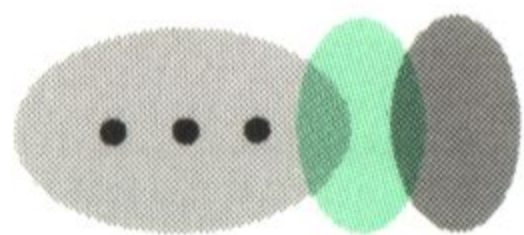
scanf｜スキャンエフ

scanf は、**printf**（プリントエフ）と似た形式の関数で、入力の働きをします。

◉ int 型変数を入れるプログラム

```
#include <stdio.h>
main()
{
    int a, b;

    scanf("%d,%d", &a, &b);
}
```

・・・scanf 文の基本書式

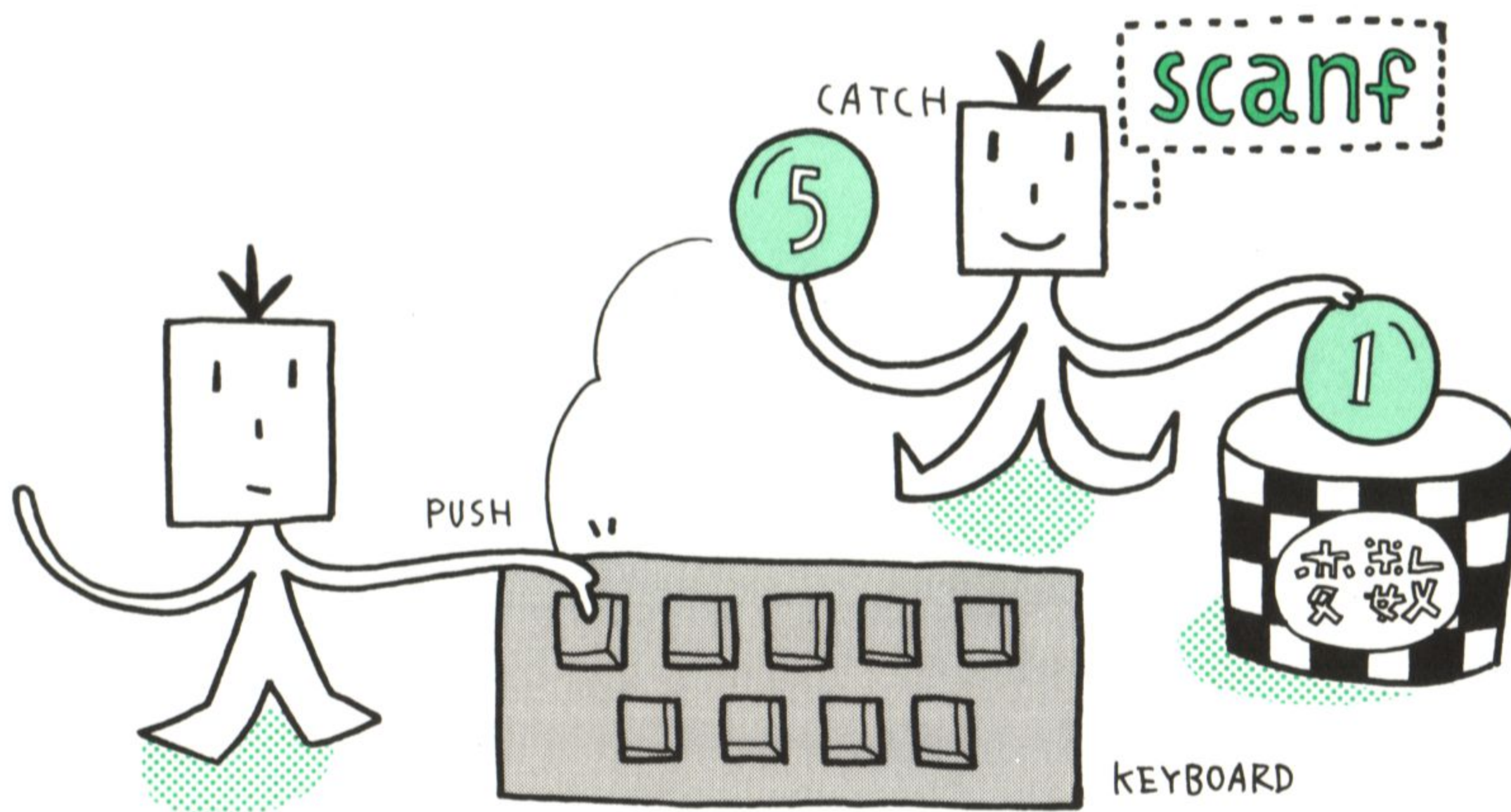
scanf（”変換文字”, &変数, &変数, ……）;

scanf 関数の「( )」の中には「” ”」で囲まれた文字と、「,」(カンマ)で区切られた変数が入っています。変数 a や b には頭に「&」記号がついているので、&a や&b は a、b とは別の変数のようですが、実は同じものです。なぜ「&」記号が必要かは理由があるのですが、それは後述する自作関数のところで説明したほうがよいので、ここでは **scanf** 文のカッコ内に並べる変数の頭には必ず「&」をつける、と覚えてください。「&」のつけ忘れは、初心者だけでなく、多くの人がおかしやすいエラーです。

・・・scanf 文の入力のしくみ

前掲のプログラムで **scanf** 文が実行されると、カーソルが点滅しながらキーボードからの入力を待ちます。

そこで、

☞ **1, 5** ⏎

のように入力すると、変数 a に 1 が、変数 b に 5 が入ります。「” ”」で囲まれているのは、変換文字「%d」と「,」です。変換文字は、**printf**（プリントエフ）文のそれと似てい

て、対応する変数に入る数字の型を示しています。「%d」は、10進数を変数に入力する働きをします。カンマが間にあるときは、複数の数字の区切りをカンマで判定するという意味です。ですから、

　　整数，整数

のように入力すると、最初の整数を変数 a に、次の整数を変数 b に入れます。

もし、このカンマがなくて、

```
scanf("%d%d", &a, &b);
```

のようになっていたらどうでしょうか？　入力したい値が10と100の場合、

☞ 10100 ↵

と入力すると、a に10100がセットされてしまいます。**scanf** をこのように使うときは、

☞ 10 ↵ 100 ↵

または、

☞ 10␣100 ↵

と、数字と数字の間に ↵ かスペースを入れて数字を区切ります。

scanf 文単独では何の文字を入力してよいかわからないので、プログラムの中で使うときには先に**printf**(プリントエフ) 文でメッセージを表示するとわかりやすくなります。

```
printf("整数を２つ入力してください¥n例 1,100¥n");
scanf("%d,%d", &a, &b);
```

↓実行結果

```
整数を２つ入力してください
例 1,100
■  ←――――カーソルが点滅して、入力を待っている
```

このメッセージを、**scanf** 文だけで表示することはできないのでしょうか？ たとえば次のような書きかたが考えられます。

```
scanf("整数を２つ入力してください%d,%d", &a, &b);
```

試してみるとわかりますが、この場合はメッセージも出ないばかりか、数字を入力しようとしてもうまく受けつけてくれません。この **scanf** 文では、次のように入力してはじめて、変数 a と b に数字10と100が入ります。

☞ **整数を２つ入力してください10,100** ↵

scanf 文の「"　"」の中に「%」で始まる変換文字以外の文字が書かれているときは、それと同じ文字を入力してようやく **scanf** が正しく動作し、変換文字のところに対応する数字や文字が入ります。ですから「"　"」の中にはカンマや変換文字のみを入れ、入力のためのメッセージは**printf**（プリントエフ） 文で表示するのです。

```
scanf("%d %d", &a, &b);
          ↑スペース
```

のように、「%d」の間にスペースが入っているとどうなるでしょう？　結論をいえば、「"　"」の中のスペースは、あってもなくても同じです。

scanf の使いかた

scanf 関数では、文字を入力するときと実数を入力するときは、入力のしかたが異なります。

6 文字の入力

まず文字を入力する場合をやってみましょう。

◉文字を入力するプログラム

```
#include <stdio.h>

main()
{
    char a, b;

    scanf("%c%c", &a, &b);
}
```

これを実行するとき、キーボードからたとえばＳとＴの文字を入力するには、次の方法ひとつしかありません。

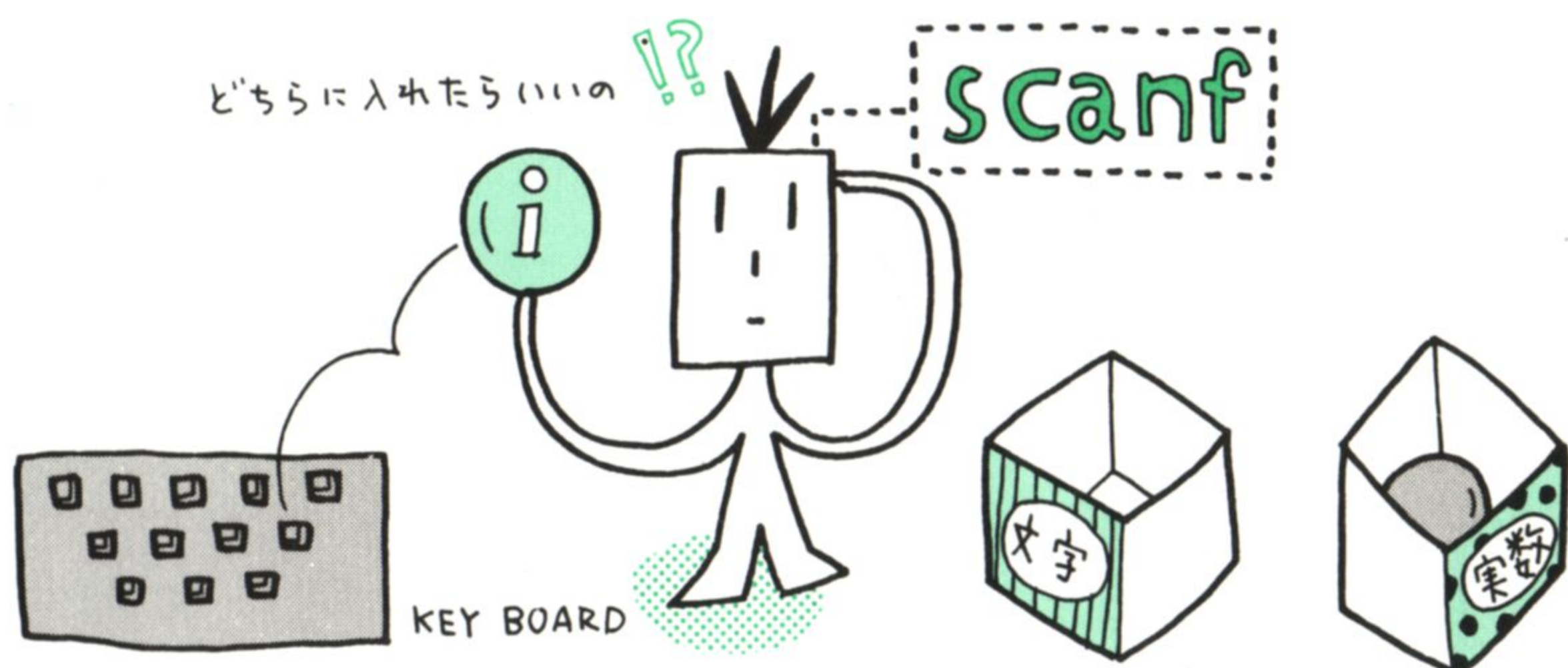
☞ ST ↵

次のような入力では、思った結果が得られません。

☞ S␣T ↵

なぜなら、これだと変数ａにＳが入り、変数ｂにはスペースが入るからです。
また、

☞ S↵T ↵

と入力すると、変数ａにはＳが入りますが、変数ｂには改行コード「¥n」が入ってしまいます。

6 実数の入力

次に、実数つまり小数点つきの数字を入力する場合です。

◉実数を入力するプログラム

```
#include <stdio.h>

main()
{
    float a, b;

    scanf("%f%f", &a, &b);
}
```

これを実行する場合は、文字の入力とは違い、次の２通りの方法があります。

☞ 1.23 ↵ 2.34 ↵

☞ 1.23␣2.34 ↵

小数点つきの数字を入力するには、このようにデータの区切りに ↵ かスペースを入力します。**scanf** 文に並べられている変数が**int**(インテジャ)型のとき、小数点つきの数字を入力するとおかしな値になりましたが、**float**(フロート)型の場合は小数点のつかない整数を入力しても、正しい値が入力されます。

また、文字型配列にデータを書き込む場合は、次のように書き表されます。

```
#include <stdio.h>

main()
{
    char a[80]

    scanf("%s", a)
}
```

１つの変数にデータを読み込む場合と違って、「&」記号をつけずに配列の名前だけを書くのが決まりです。これを実行するには、たとえば「Good morning」のように文字列を入力すると、スペースまたは ↵ の前までの文字が変数に読み込まれ、最後に「¥0」がつけられます。

getchar ゲットキャラクタ

getchar 文の基本書式

getchar 関数は、変数に文字を1字入力する関数です。

```
変数 = getchar( );
```

このように書くと、変数に、キーボードから入力された文字が1字入ります。この場合の変数は、**int**(インテジャ)型でなければなりません。

getchar の使いかた

◉入力した文字数をかぞえるプログラム

```
#include <stdio.h>
/* 半角文字を入力してその数をかぞえる */

main()
{
    int x, i = 0;        ←整数型の変数宣言と初期値の設定

    while((x = getchar())!= '¥n') i++;    ←ループ

    printf("¥n%d文字入力しました", i);
}
```

↓実行結果

```
■          ←カーソルが点滅している
↓
asdf12ZXC ↵ ←入力
↓
asdf12ZXC         ┐
9文字入力しました ┘←表示
```

このプログラムを実行すると、画面にはカーソルが点滅して入力を待っている

状態になります。そこで、キーボードの文字や数字のキーをいくつか押してから、最後に⏎キーを押します。そうすると、続いてprintf(プリントエフ)文のメッセージが表示されます。

while(ホワイル)文のところがむずかしいですね。

```
while((x = getchar())!= '¥n') i++;
```

カッコ内の式が複雑なので、わかりにくくなっています。この1行を分解して書いてみましょう。

```
x = getchar();
while (x != '¥n') {
    i++;
    x = getchar();
}
```

まず、**getchar** 文で入力した文字が変数 x に入ります。これが「¥n」（改行コード）でないかどうかを判定してそれ以外の場合はループに入り、i を 1 増やします。次に、また x に新しい文字が入り、という繰り返しです。

上の 4 行をまとめて書くと、

```
while((x = getchar())!= '¥n') i++;
```

になるのです。

このように省略して書く方法は、C 言語ではよく行われます。慣れないと、プログラムの意味を理解するのに時間がかかってしまいます。

疑問の点がもうひとつあります。キーボードからの入力は 1 回なのに、

```
x = getchar();
```

で x に文字を代入する式は何度も繰り返されます。なんだかへんな感じですが、実は **getchar** は、入力バッファから文字を読み込む働きをする関数なのです。入力バッファとは、キーボードから入力された文字などを、一時的にためておく場所のことです。

入力バッファに文字が何もないときは、キーボードからの入力待ちになります。入力が行われ、最後に⏎キー、すなわち「¥n」コードが読み込まれると、入力は終了します。そして、バッファに文字がある間はここから文字を読み込みます。

x = getchar( ) ;

キーボードからの入力
「¥n」を入力して終了

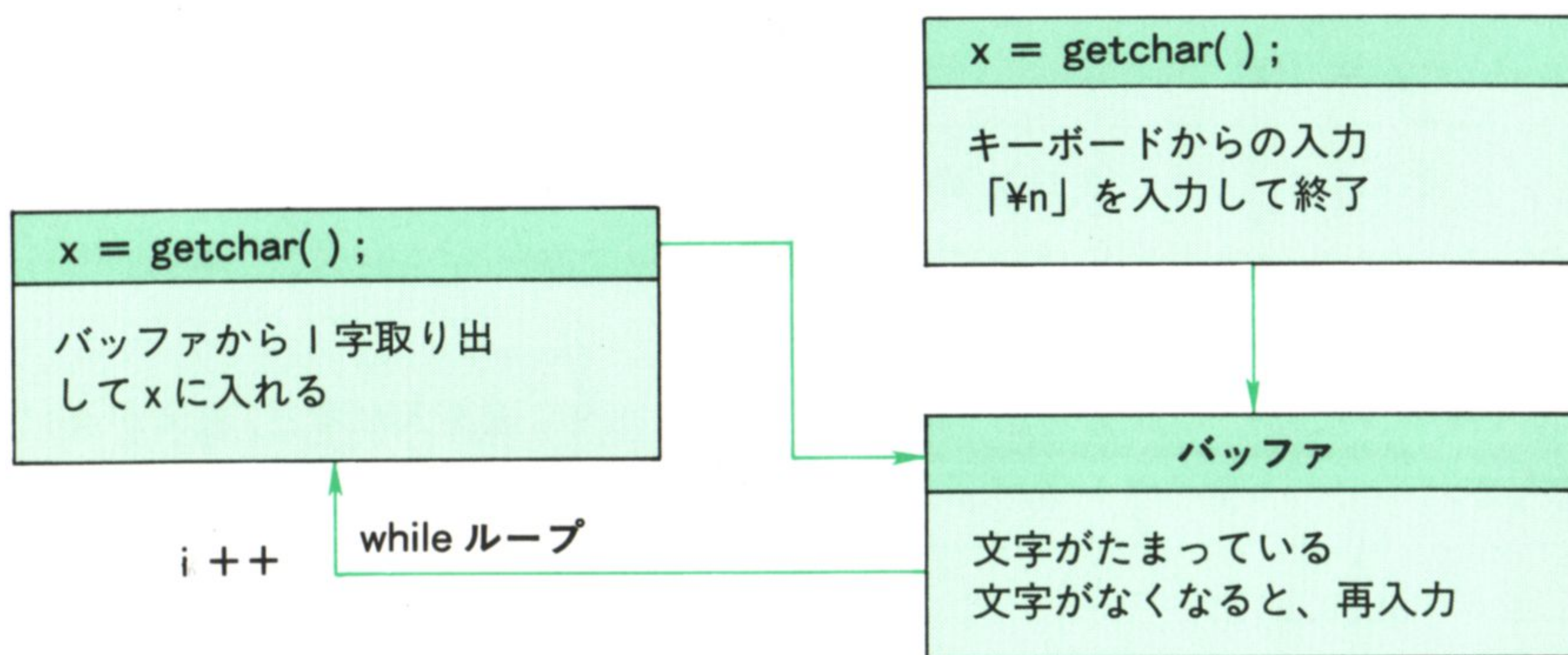
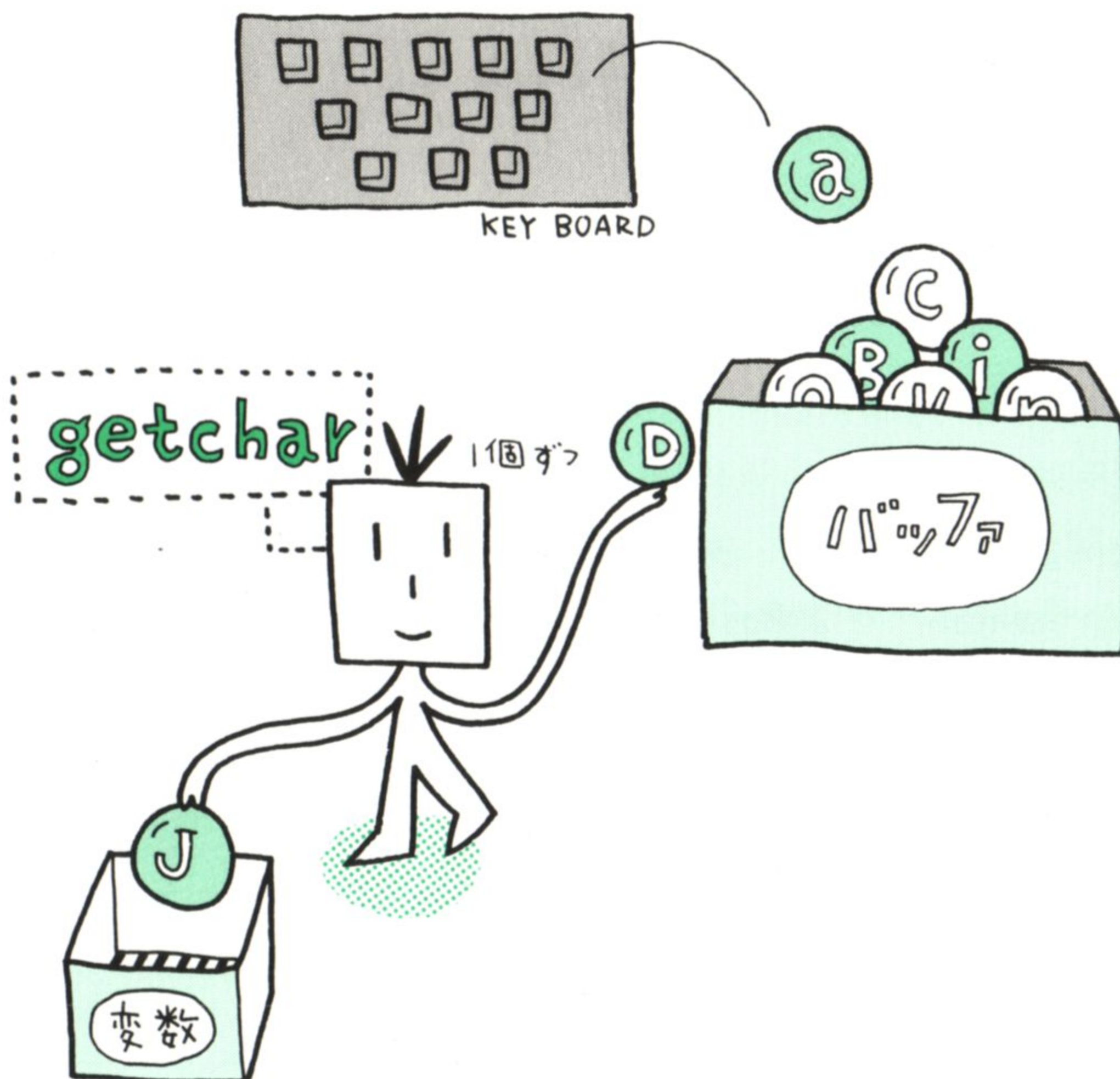
x = getchar( ) ;

バッファから1字取り出
してxに入れる

バッファ

文字がたまっている
文字がなくなると、再入力

while ループ

i ++

図のように、**while**文が1回まわるたびに、バッファの文字が1個ずつ取り出されてxに入ります。これを、iでカウントするのです。

そこで、前掲のプログラムに話を戻すと、**while**ループに入るときに**getchar**文が実行され、変数xに1文字代入されます。その値が「¥n」でない場合、iに1がプラスされます。「¥n」すなわち最後の改行コードのところへくるとループから抜け、それまでカウントした文字数を表示するわけです。

自作関数の作りかた、使いかた

関数作りの基本

プログラムが大きくなると、「main ( )」ひとつで書くのはめんどうですし、プログラムが複雑になるので、関数として分けるほうが便利です。決まった働きをする部分を関数として自作し、使うときはmainの中でそれを呼び出すようにするわけです。「main( )」が2画面（50行）以上になったら関数を作る、というのもひとつの目安になるでしょう。

ここでは、関数の作りかたと関数の性質、いろいろな約束ごとについて説明していきます。

関数は、たとえば自動販売機のようなものです。

お金を入れてボタンを押すと品物が出てくる

のが自動販売機ですが、関数もこれと同じようなもので、

y = f(x)

という形をしています。数学の関数には $y = \sin x$ や $y = x + \frac{1}{x} + 1$ のようなものがあり、これも意味としては同じです。つまり、xに何かを入れると、その結果を返す、というのが関数の働きです。

C言語では、たとえば次のように書かれ、カッコ内に値を入れると、答えがxに代入されるという働きをします。

```
x = getchar();
fp = fopen(name, "r");
```

C言語の関数は、カッコ内に何も書かれない場合もあれば、左辺つまり代入する変数を省略する場合もあります。

```
printf("a = %d", a);
```

いろいろ例外はありますが、まず最も基本的な関数を作ってみましょう。

関数作りの実際

和暦の年号を入れると、西暦に変換してくれる関数を作ります。関数の名前は、「nengou」とします。年号と何年かという数字を入力するわけですが、年号は次のように記号（コード）化して、1文字を入力すればよいようにしておきます。

明治――m
大正――t
昭和――s
平成――h

◉年号を西暦に変換する関数――1

```
int nengou(a, b)
char a;
int b;
{
    int c = 0;

    switch(a) {
        case 'm':
            c = b + 1867;        /* 明治を西暦に変換 */
            break;
        case 't':
            c = b + 1911;        /* 大正を西暦に変換 */
            break;
        case 's':
            c = b + 1925;        /* 昭和を西暦に変換 */
            break;
        case 'h':
            c = b + 1988;        /* 平成を西暦に変換 */
            break;
        default:
            c = -1;
            break;
    }

    return c;
}
```

この関数は、aに年号に対応する文字コード、bに年数をセットすると、西暦を返します。変数aがmのときは明治、tのときは大正、sのときは昭和、hの時は平成をそれぞれ計算し、どれにも当てはまらないときは結果を「－1」にします。

例として、「昭和45年」を西暦に変えたいときは、次のように記述します。

```
a = 's';      ←――――昭和
b = 45;       ←――――45年
c = nengou(a, b);
```

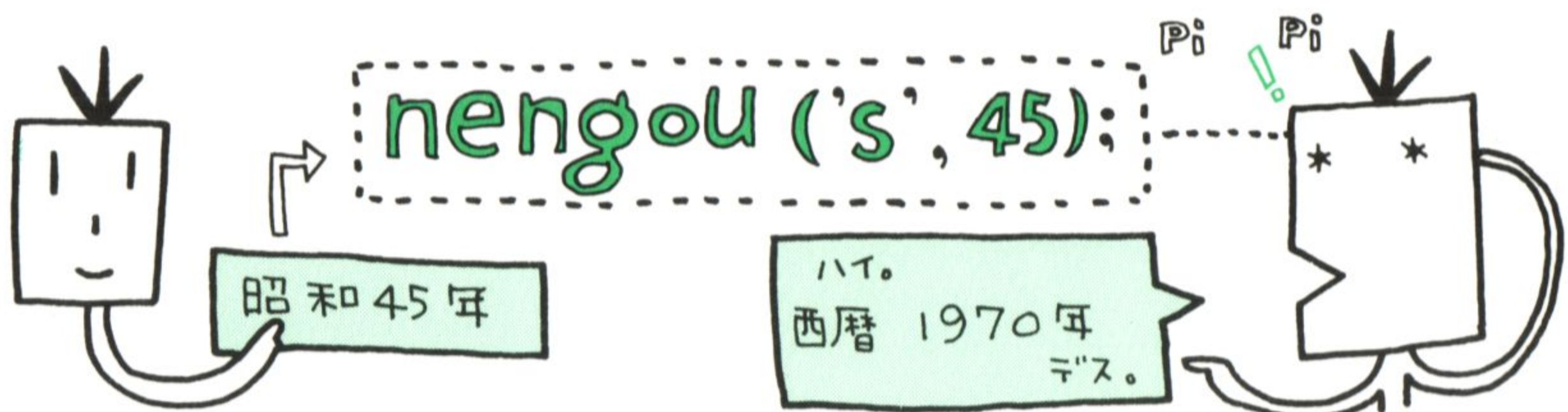
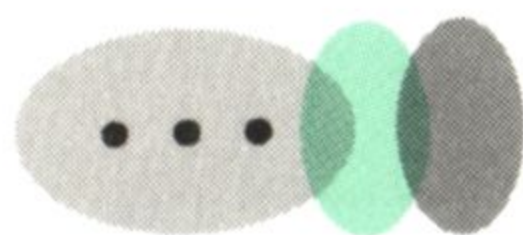
または、直接書き込むこともできます。

```
c = nengou('s', 45);
```

文字を直接入れる場合は、必ず「'　'」ではさんでください。これは、変数に文字定数を代入する場合と同じです。数字は、直接書くことができます。

メインプログラムから、この関数を呼んでみましょう。

◉「昭和45年」を西暦に変換するプログラム

```
#include <stdio.h>

main()
{
    char n;
    int l, m;

    n = 's';
    l = 45;
    m = nengou(n, l);     ←――――関数の呼び出し

    printf("西暦%d年です", m);
}
```

関数を作る約束ごと

さて、この関数プログラムとメインプログラムの例を見ながら、関数を作るときの約束ごとをチェックしていきましょう。

6 関数名と引数（ひきすう）

```
int nengou(a, b)
char a;
int b;
{
```

関数プログラムの最初の行には、関数の名前が書かれます。「int」とあるのは、この関数の値が**int**型（インテジャ）だという宣言です。関数の値というのは、メインプログラム中に、

```
m = nengou(n, 1);
```

と書いたときの m に入る値で、返値（へんち）という表現をすることもあります。

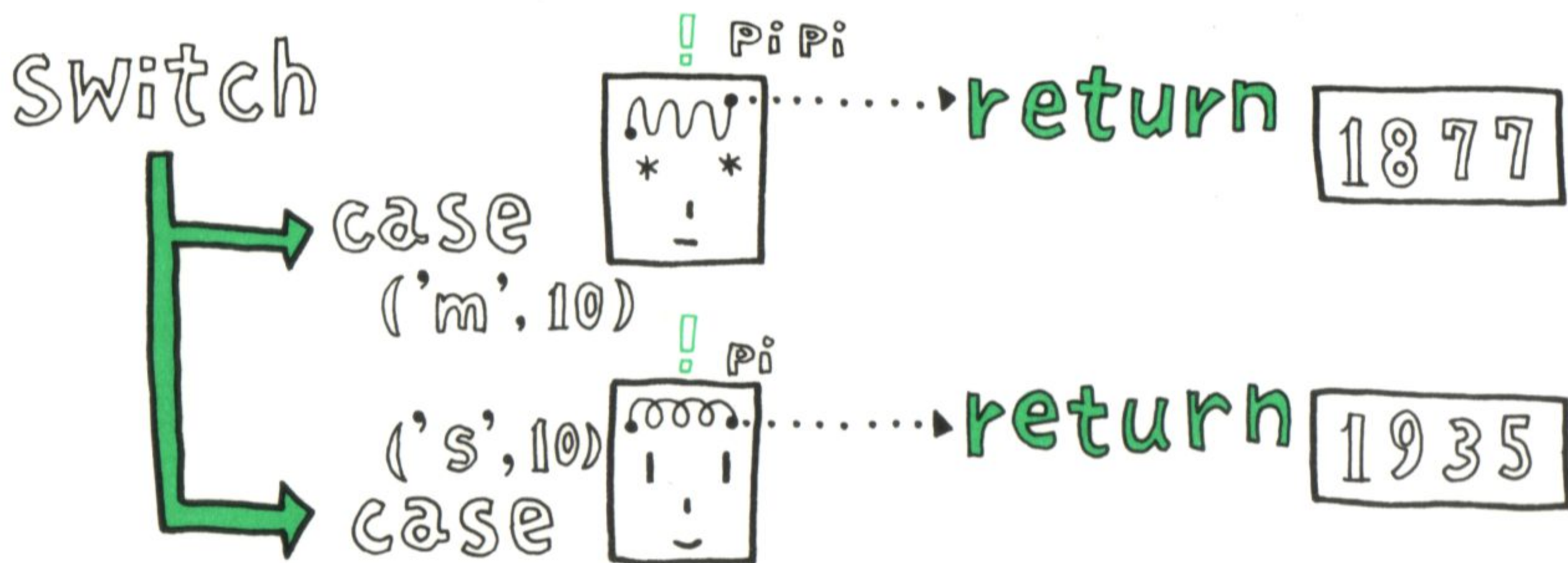
関数名の次のカッコ内に、変数 (a, b) が入っています。これは引数（ひきすう）と呼ばれます。nengou の引数は 2 つですが、3 つでも 4 つでも、場合によっていくつでも入れることができます。

関数プログラムの 2 行め以下が、引数の型宣言です。

```
char a;
int b;
```

引数は、必ずここで型宣言して使います。宣言しなかったり、別の場所で宣言してはいけません。この例では a に**char** 型（キャラクタ）、b に**int** 型（インテジャ）の変数を入れます。関数を呼ぶときに入れる変数は、関数の中で使った変数と異なっていてもかまいません。この例のように、関数のほうでは「int nengou(a,b)」と宣言しているのに、関数を呼ぶときは「m = nengou(n, l);」と、別の変数を入れてもさしつかえないのです。

6 関数の本体とreturn（リターン）文

関数プログラムの「{」からが、関数の本体です。関数の中だけで使われる変数は、「{」の次に宣言されます。

和暦を西暦に変える計算を行っているのは、**switch～case**文です。各**case**文の最後に**break**文がついているので、**case**文に分岐したら「}」を抜けます。

次に、

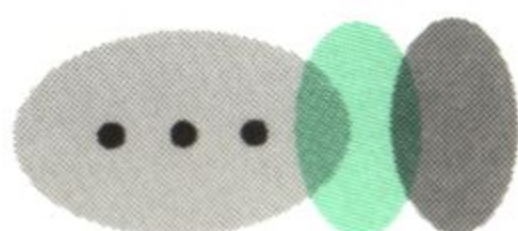
```
return c;
```

があります。ここに書かれた値 c が、関数の値（返値）となります。**return**文があると、関数はそこで終わります。ですから、**break**文の位置に**return**を書くと、そこで終わって main に戻ります。**break**文が必要ないわけですから、プログラムはさらに簡略になります（下の例参照）。

関数にはいくつもの値を渡すことができますが、返される値は1つです。2つ以上の値を返すときは、グローバル変数とするか、次項で述べるポインタを使うという2つの方法があります。これらは少しむずかしくなりますから、ここでは触れません。

◉年号を西暦に変換する関数――2

この位置に return 文を使用

```
int nengou(a, b)
char a;
int b;
{
    switch(a) {
        case 'm': return (b + 1867);    /* 明治を西暦に変換 */
        case 't': return (b + 1911);    /* 大正を西暦に変換 */
        case 's': return (b + 1925);    /* 昭和を西暦に変換 */
        case 'h': return (b + 1988);    /* 平成を西暦に変換 */
        default:  return (-1);
    }
}
```

関数とメインプログラムのまとめかた

プログラムで使う関数は、メインプログラムと同じファイルとしてまとめます。上記の例では、関数 nengou と main を同じファイルに入れておくわけです。main だけあっても、nengou がないとプログラムは実行できませんし、main のほうがなくても同じことです。

必要な関数が足りない場合、プログラムをコンパイル→リンク→実行しようとすると、リンクの際にエラーが出てストップします。

```
C>link s¥c b:nengou ,b:nengou,,lcm lc

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985


Unresolved externals:


NENGOU in file(s):
 B:NENGOU.OBJ(NENGOU)

There was 1 error detected
```

▲エラーメッセージの例

このエラーはコンパイル時のものではないので、何が原因かがわからず、困ってしまう人が多いようですが、関数が足りないために起きるエラーです。関数は、忘れずにすべて1つのファイルに入れておきましょう。

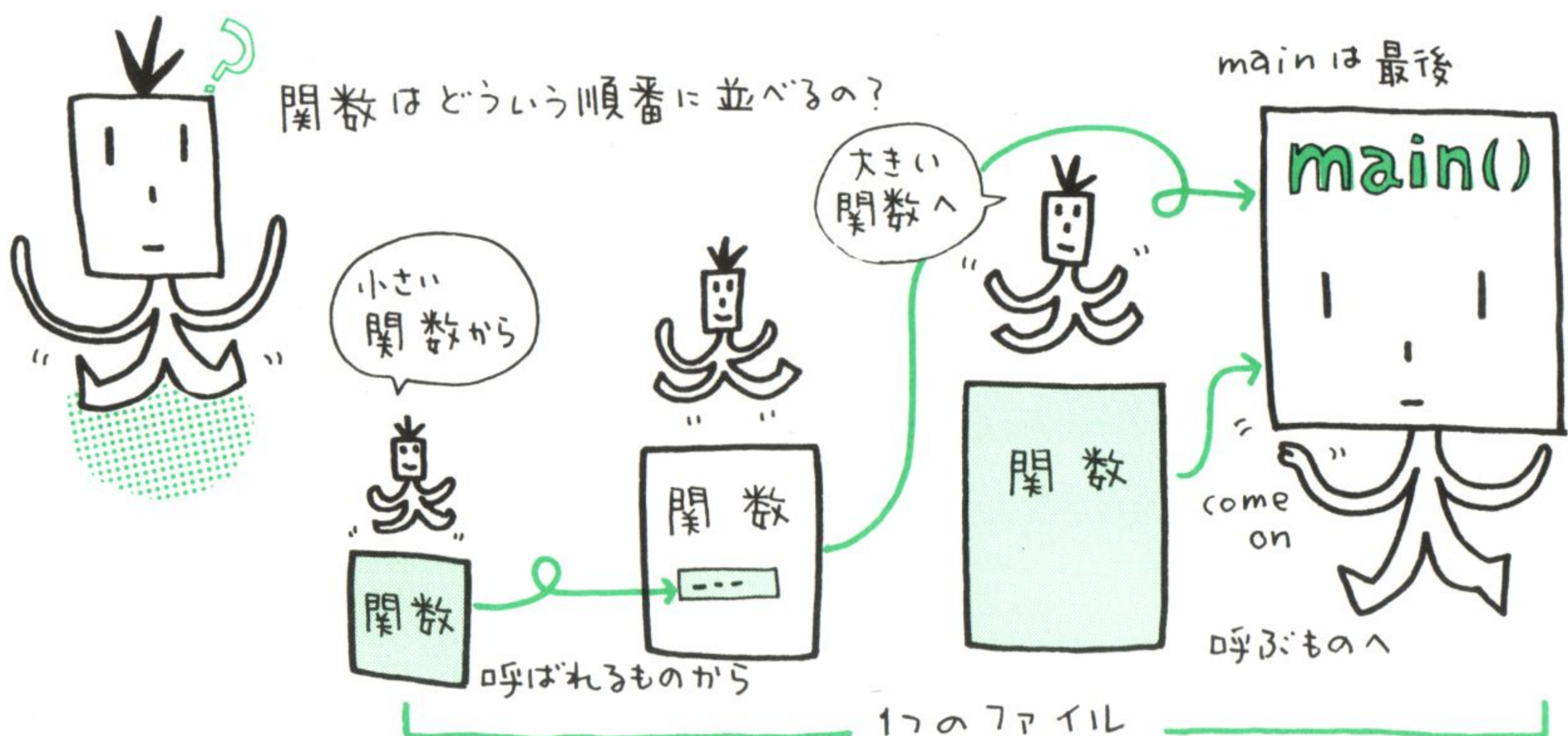
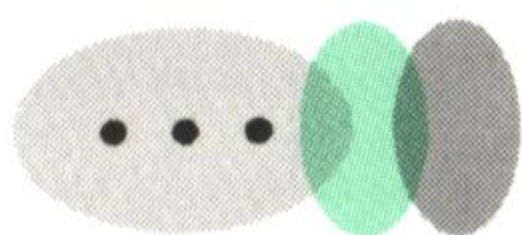
プログラムが大きくなると、1つのファイルには入りきらず、複数のファイルに分けて入れるようになりますが、初心者のうちはおそらくプログラムも小さいので、1つのファイルにまとめる、と考えておけば十分です。

では、1つのプログラムの中で多くの関数を使うとき、どういう順に並べてまとめるかが、次に問題となります。通常は、小さい関数から大きい関数へ、呼ばれるものから呼ぶものへ、そして main はいちばん最後に書くようにしてください。このことは初心者向けの本ではあまり書かれていませんが、プログラミングのコツのひとつです。この方法がよい理由は「main やほかの関数から関数を呼ぶとき、初めに関数の型宣言を全部行っていないとコンパイルエラーとなる」からです。

次の例では、func 1、func 2 の 2 つの関数があり、func 2 の中で func 1 を、そして main の中で両方の関数を呼んでいます。

◉複数の関数を使う

```
char func1()  ←――関数1の型宣言。関数は呼ばれるほうが先、呼ぶほうがあと
{
    ……
}
float func2()  ←――関数2の型宣言
{
    ……
    a = func1();  ←――関数2の中で関数1を呼んでいる
    ……
}

main()  ←――最後にメインを書く
{
    x = func1();  ┐
    ……           ├―関数1、2を呼ぶ
    y = func2();  │
    ……           ┘
}
```

実際に多くの関数を使って大きなプログラムを作っていく場合は、先頭でまとめて全関数の型宣言を行えばよいのですが、あとから関数をつけ足したり、いろいろ手を加えているうちに、必要な型宣言を忘れることもしばしばです。いろいろな場所で呼ばれる関数を先頭に、比較的大きめの関数を後ろに、main を最後に書くようにすると、関数を呼ぶときにはすでに関数の型が宣言されているため、問題はなくなります。

値のない関数

関数の返値は1つだけですが、必要がなければ値を省略することもできます。値のない関数は、**void**(ボイド)型として宣言することができます。

「void」とは、形がないとか、不定型なというような意味です。プログラムに必ず1つ記述するmainは、値のない関数の代表的な例です。そこで、これをあらかじめ、

```
void main()
```

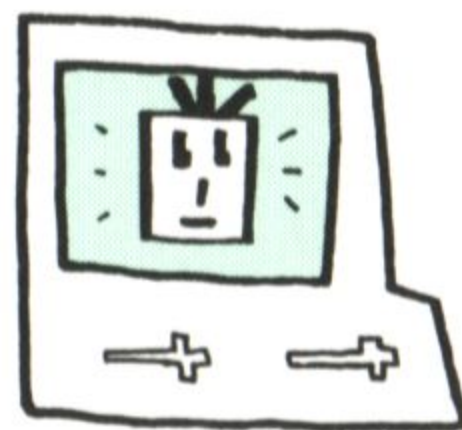
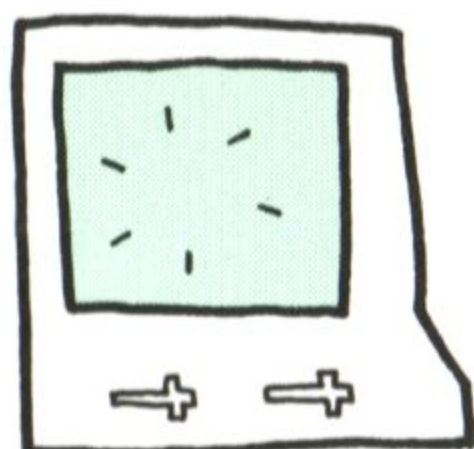
と書くように習慣づけるとよいでしょう（void型は、1989年に規定されたANSI＝米国国家規格協会の規格から正式に取り入れられた）。

値のない関数の例を1つ紹介しておきます。これは、時間を少し遅らせるという関数で、名前はtimeとします。この関数は、たとえば画面に何かを表示するとき、一瞬のうちに消えてしまわないよう、時間を少し遅らせるというような場合に使います。このような値のない関数を呼ぶには、

```
time();
```

のように、関数名をそのまま書くだけでよいのです。

◉値のない関数の例

```
/* 時間を遅らせる関数 */

void time()
{
    int time;

    for ( time = 1; time < 32768; time++ ) ;
}
```

ポインタの働きとしくみ

ポインタの働き——int 型

ポインタは、通常の高級言語であるBASIC(ベーシック)やFORTRAN(フォートラン)、COBOL(コボル)などにはないC言語特有の機能です。プログラム中で用いられ、いろいろと便利な働きをします。

ポインタとはもともと英語で、「pointer＝指さすもの」という意味です。何を指さしているかはいろいろで、文字を指すもの、整数を指すもの、実数を指すものなどがあります。

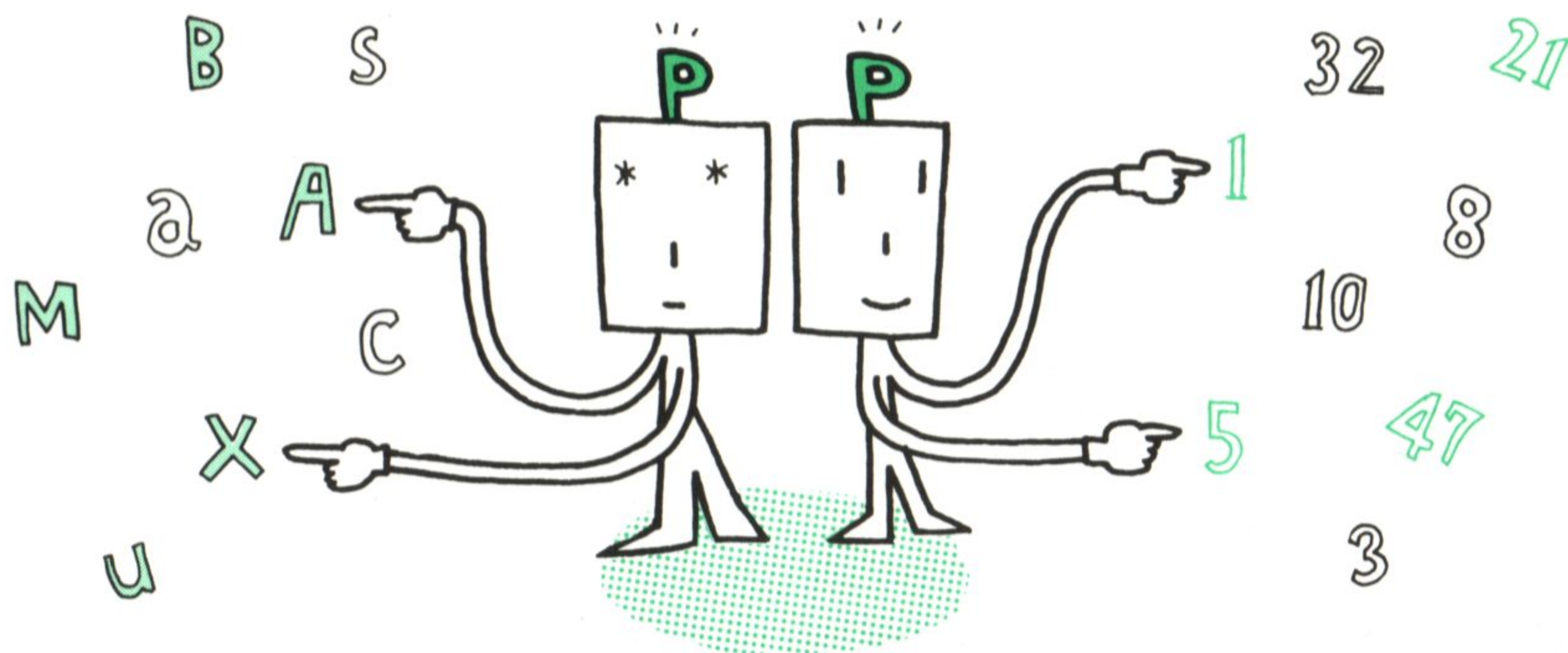
ポインタの種類によって、文字を指すポインタは文字ばかり、整数を指すポインタは整数ばかりというように、ポインタにもタイプがあって種類が分かれているのです。実際にどういうものかは、プログラム例で見ていきましょう。

◎ int 型のポインタを使ったプログラム——1

```
#include <stdio.h>

main()
{
    int *a;
    int x;

    x = 1000;       /*  xに1000を入れる  */
    a = &x;         /*  xのアドレスをaに入れる  */

    printf("*a = %d", *a);    /*  aの指し示す中身は？  */
}
```

↓実行結果

```
* a = 1000
```

実行結果の表示を見ると、どうも「＊ a」には変数 x と同じものが入ったようです。この場合、プログラムの中で、

```
int *a;
```

と宣言されているaがポインタです。次の行のint（インテジャ）型変数xとは違い、「＊」記号がつけられています。このaは「int型のポインタ」といわれるものです。

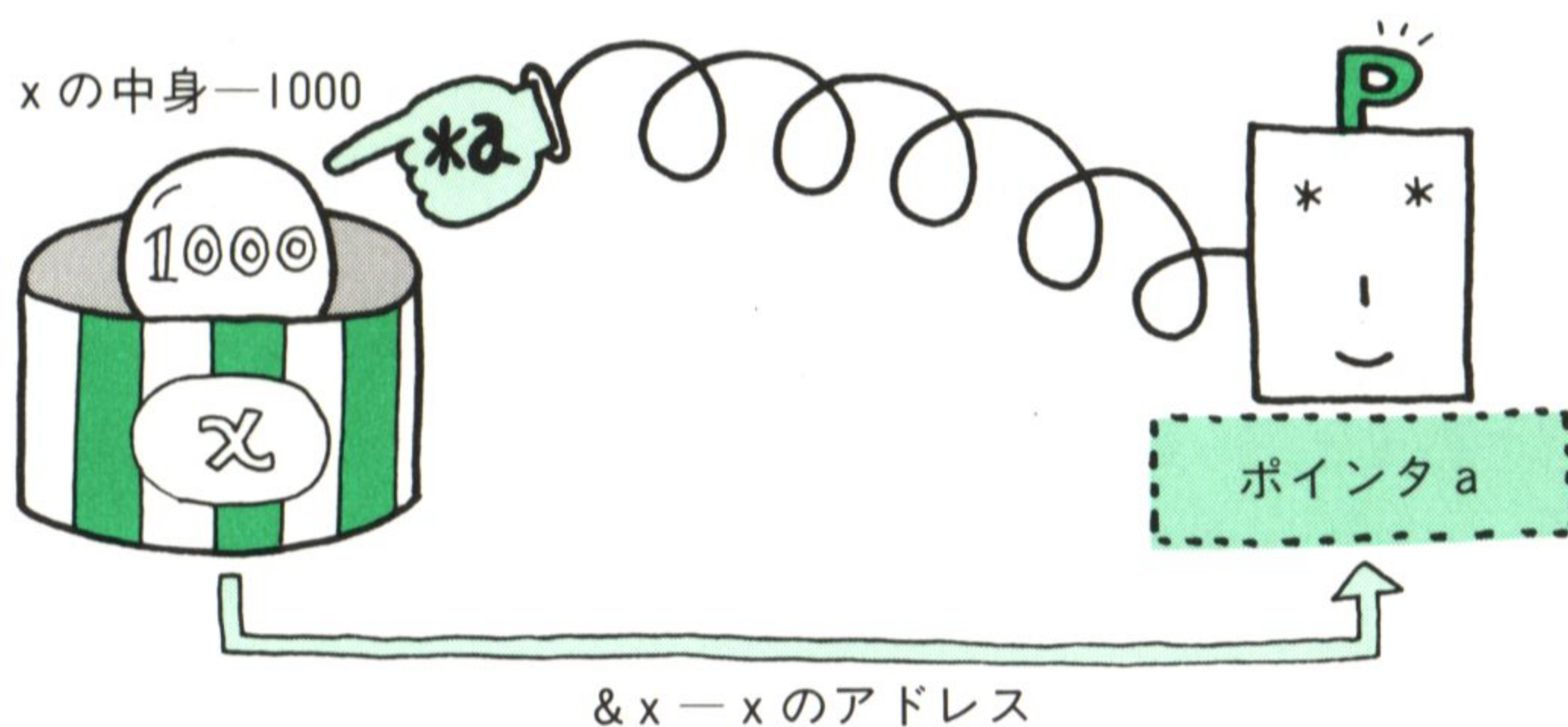
では、このポインタはどのように使われているのでしょう。

まず、

```
x = 1000;
```

でxに整数1000が代入されます。つまり、変数xの箱の中に1000が入れられたということです。

次の行には、見なれない「&x」があります。この「&」（アンパサンド）は演算子の一種で、変数のある場所、すなわちxのアドレスを示します。ですからこの式は、ポインタaにxのアドレスを代入したことになります。

そして「＊a」に戻ります。「＊」はポインタの指しているアドレスの中身を表します。

xの中身―1000

&x ― xのアドレス

ポインタa

そこで、**printf** 文が実行された結果、「＊ａ ＝ 1000」が表示されたわけです。

このプログラムで使われているポインタａは、実は**int** 型変数のアドレスを入れる箱です。この箱の中にｘのアドレスを入れたため、＊ａを表示させるとｘの中身1000が表示されました。

前出の**scanf** 文で変数の頭につけられていた「&」記号も、この場合と同じ働きをする演算子です。入力された数字などをプログラムの中でいろいろと操作するとき、そのアドレスを把握しておく必要があるわけです。

前出のプログラムを、少し変更してみましょう。

◉ int 型のポインタを使ったプログラム——２

```
#include <stdio.h>

main()
{
    int *a;
    int x;

    x = 1000;         /*  xに1000を入れる  */
    a = &x;           /*  xのアドレスをaに入れる  */
    printf("*a = %d¥n", *a);    /*  aの指し示す中身は？  */

    x = x + 1000;       /*  xに1000を足す  */
    printf("*a = %d", *a);    /*  aの指し示す中身は？  */
}
```

↓実行結果

```
＊ａ =1000
＊ａ =2000
```

ポインタａに変数ｘのアドレスを入れ、最初の中身「1000」を表示するところまでは前出のプログラムと同じです。

この場合は、その後にｘの値を変更しましたが、それでも＊ａは変更後のｘの値を表示しています。つまり、ポインタａがｘのアドレスの中身を指し示すことは変わりませんから、その中身であるデータをいくら操作しても、ポインタは正しくそれを示すわけです。

今度は、次のようにプログラムを変更してみましょう。

◉ int 型のポインタを使ったプログラム——3

```
#include <stdio.h>

main()
{
    int *a;
    int x;

    x = 1000;        /*  xに1000を入れる  */
    a = &x;          /*  xのアドレスをaに入れる  */
    printf("*a = %d¥n", *a);    /*  aの指し示す中身は？  */

    *a = *a + 1000;             /*  *aに1000を足す  */ ←変更部分
    printf("*a = %d", *a);      /*  aの指し示す中身は？  */
}
```

↓実行結果

```
*a =1000
*a =2000
```

このようにしても、結果は同じになります。つまり、x の代わりに＊ a と書いても同じ意味になるのです。

float 型、char 型のポインタ

◉ float 型のポインタを使ったプログラム

```
#include <stdio.h>

main()
{
    float *a;
    float x;

    x = 3.14159;        /*  xに3.14159を入れる  */
    a = &x;          /*  xのアドレスをaに入れる  */

    printf("*a = %f", *a);     /*  aの指し示す中身は3.14159  */
}
```

int（インテジャ）型の例でポインタを説明してきましたが、**float**（フロート）型、**char**（キャラクタ）型でも同様です。プログラム例と実行結果だけを紹介しておきますから、その働きを確認してください。

↓実行結果

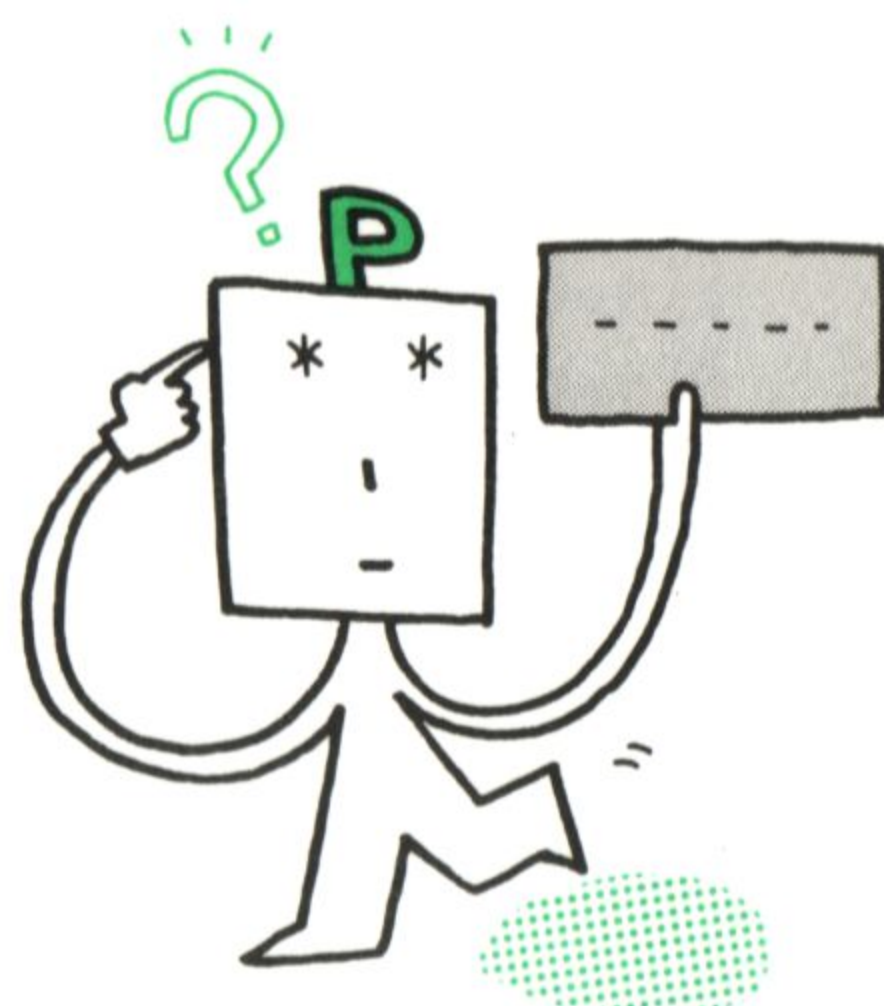
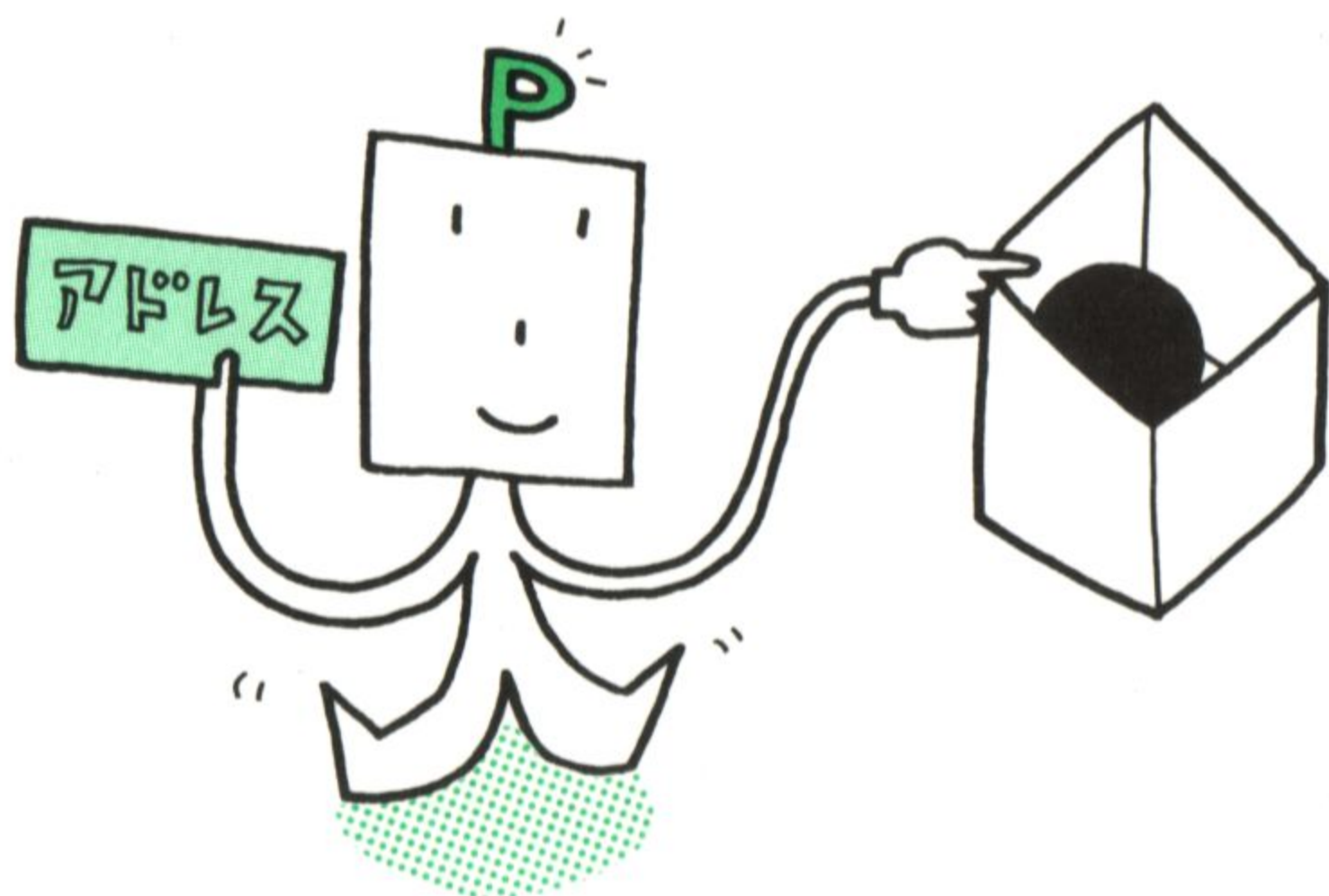
```
* a =3.141590
```

◉ char 型のポインタを使ったプログラム

```
#include <stdio.h>

main()
{
    char *a;
    char  x;

    x = 'A';                /*  xに文字Aを入れる  */
    a = &x;                 /*  xのアドレスをaに入れる  */

    printf("%c", *a);       /*  aの指し示す中身は'A'  */
}
```

↓実行結果

```
* a= A
```

ポインタにはアドレスが入ります。ポインタが指しているのは、そのアドレスにあるデータです。ですから、ポインタに何も入っていないとき、ポインタは何も指し示しません。このことは覚えておいてください。

構造体のしくみと考えかた

構造体のしくみ

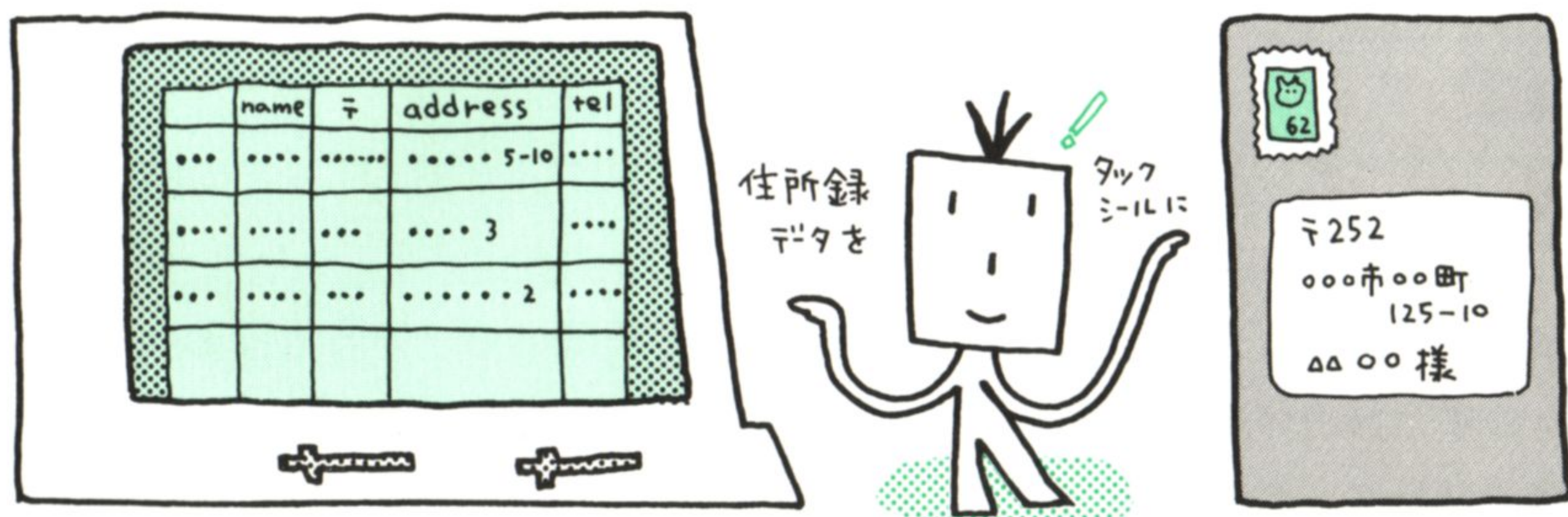
構造体というとなにやらいかめしい感じですが、これも一種の変数だと思えばわかりやすいかもしれません。

構造体は、データベースを例にすると、簡単に理解できます。データベースはひとまとまりのデータを登録するもので、初めにデータの定義ということを行います。たとえば「住所録」のデータベースを作るときは、データの項目をまず次のように設定します。

ふりがな
氏　　名
郵便番号
住　　所
電話番号

そして、設定したそれぞれの項目の属性、すなわち数字か文字か、必要な字数、といったことを指定して定義とします。この定義されたひとまとまりの項目が、データベースの1レコード（1件のデータ）になります。実際の住所録データベースでは、1つひとつの項目を順に登録していくことになります。

ふりがな	氏　　名	郵便番号	住　　　　所	電話番号
⋮	⋮	⋮	⋮	⋮

構造体の働きは、この1レコードと実によく似ています。ここでは、簡単な構造体の例をとりあげることにしましょう。4科目のテスト結果を入れる構造体を定義します。

◉テストの成績を管理する構造体

```
struct test
{
    char name[20];
    int kokugo;
    int sansu;
    int rika;
    int shakai;
    int gokei;
};
```

struct（ストラクチャ）は構造体の定義を行うステートメントで、ここではtestという名前の構造体を定義しています。test構造体には1つのレコードに、1つの配列と5つの変数が含まれていて、これらが全体で一組の変数のようにふるまいます。また、このそれぞれの変数を、構造体のメンバーと呼びます。

char型の配列――name［20］；
int型の変数――kokugo;
int型の変数――sansu;
int型の変数――rika;
int型の変数――shakai;
int型の変数――gokei;

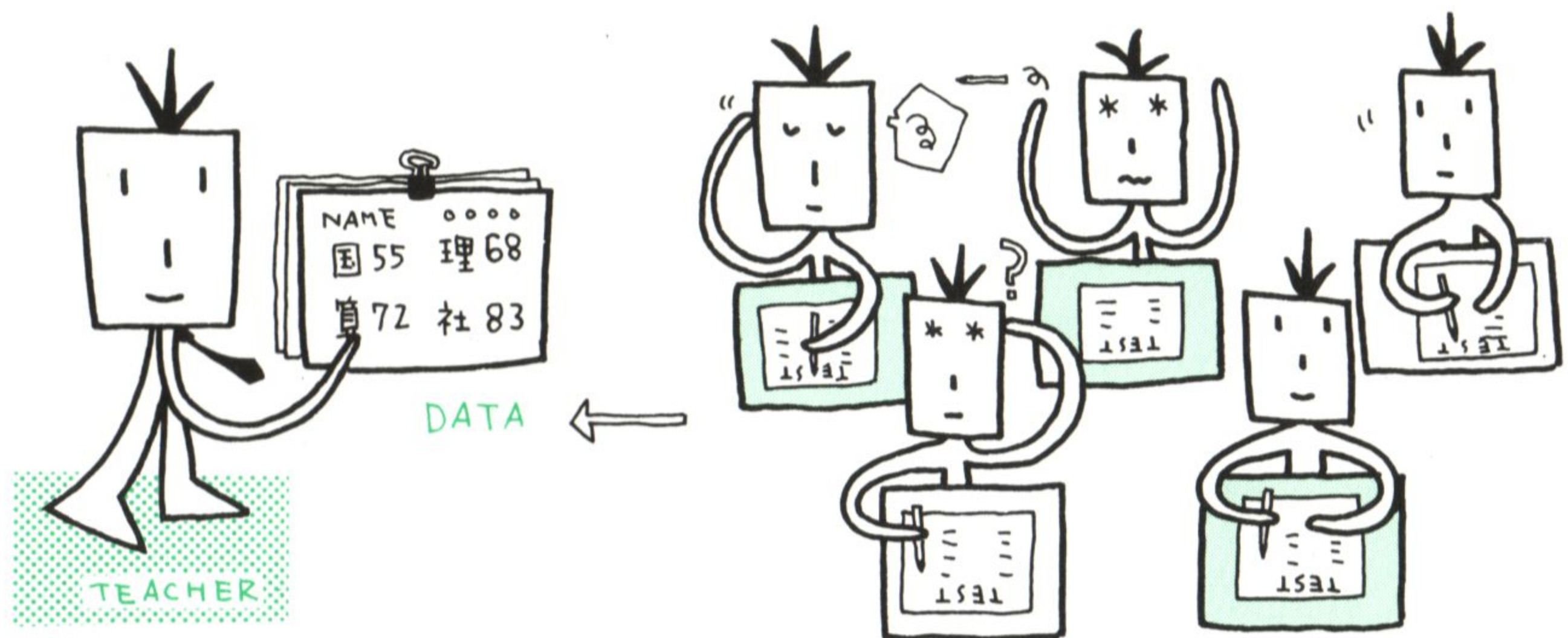
上述の構造体は、形を定義しただけです。実際に使う際には、さらに構造体変数というものを定義する必要があります。1つの構造体を定義しておくと、同じ形式をした構造体変数をいくつも使うことができます。

構造体変数は、次のように定義します。

```
struct test test1;
```

同様に、

```
struct test test1, test2, test3;
```

などのようにtest 1、test 2、test 3をまとめて定義することもできます。プログラムの中で構造体変数を使うときは、構造体変数名にピリオドをつけて、次に構造体のメンバー名を続けます。

```
test1.name
test1.kokugo
test1.sansu
……
```

構造体の使いかた

構造体の定義だけでは、実体がよくわからないでしょう。実際のプログラムで、使われかたを見ることにします。

◉構造体を使ったプログラム

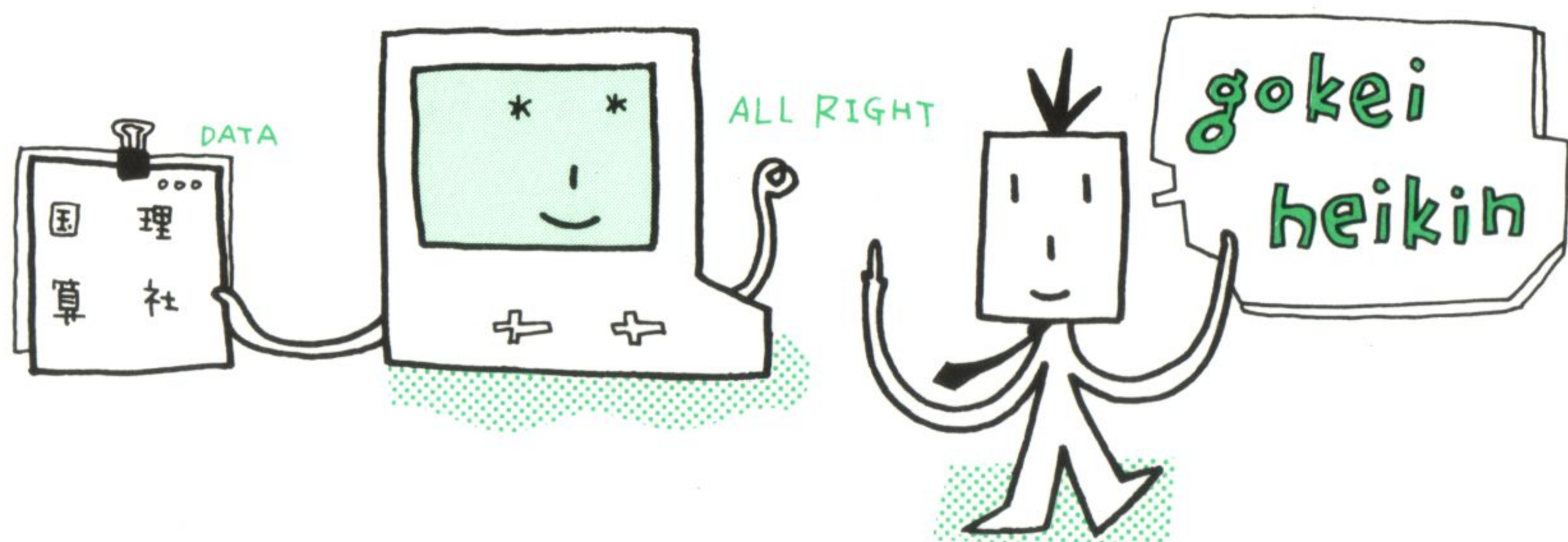
```
#include <stdio.h>

struct test
{
    char name[20];
    int kokugo;
    int sansu;
    int rika;
    int shakai;
    int gokei;
};
struct test test1[100];  /* 100人ぶんのテストデータを入れる */

/* 構造体変数test1に100人ぶんのデータを入力する */
int input_test1()
{
    int i, num = 0;
```

構造体変数の定義

```
    printf("テストのデータを入力してください¥n");

    for ( i = 0; i < 100; i++ ) {
        printf("名前を入力してください：");
        scanf("%s", test1[i].name);

        if ( test1[i].name[0] == 'Q' ) break;
            /* 名前の代わりにQを入力するとそこでループが終了 */
        printf("国語の点数を入力してください：");
        scanf("%d", &test1[i].kokugo);
        printf("算数の点数を入力してください：");
        scanf("%d", &test1[i].sansu);
        printf("理科の点数を入力してください：");
        scanf("%d", &test1[i].rika);
        printf("社会の点数を入力してください：");
        scanf("%d", &test1[i].shakai);
        num++;
    }

    return num;                         /* 入力したデータの数 */
}


/* 各人の四科目の合計点を計算する */
void gokei_test1(num)
int num;
{
    int i;

    for ( i = 0; i < num; i++ ) {
        test1[i].gokei = test1[i].kokugo + test1[i].sansu
                        + test1[i].rika + test1[i].shakai;
        printf("%s の合計点は %d 点¥n", test1[i].name,
                                        test1[i].gokei);
    }
}

/* テストを受けた人全体で四科目の平均点を計算する */
void heikin_test1( num )
int num;
{
```

```
    float a = 0.0, b = 0.0, c = 0.0, d = 0.0;
    int i;

    for ( i = 0; i < num; i++ ) {
        a = a + test1[i].kokugo;
        b = b + test1[i].sansu;
        c = c + test1[i].rika;
        d = d + test1[i].shakai;
    }

    a = a / num;
    b = b / num;
    c = c / num;
    d = d / num;

    printf("国語の平均点は %6.2f点¥n", a);
    printf("算数の平均点は %6.2f点¥n", b);
    printf("理科の平均点は %6.2f点¥n", c);
    printf("社会の平均点は %6.2f点¥n", d);

}

main()
{
    int num;

    num = input_test1();
    gokei_test1(num);
    heikin_test1(num);
}
```

構造体test1を最初に定義し、それを以下の各ブロックで変数として利用したプログラムです。input_test1で100人ぶんのテストデータを入力し、gokei_test1で各人のテストの合計点、heikin_test1で各科目ごとの平均点を計算して結果を表示します。

構造体変数test1は、どの関数からも呼べるように、プログラムの先頭に定義しています。このように、プログラムの先頭で、mainやすべての関数の外の位置に定義されている変数を、グローバル変数と呼びます。これに対して、宣言されたブロック内でだけ有効な変数を、ローカル変数と呼びます。グローバル変数は、

プログラムのどこからでも呼べるので、プログラムの全体を通して使用するデータに用いると便利です。

構造体変数 test1 は、次のような順序で並ぶ100個のレコードからなる配列です。この100個の構造体変数を利用して、4科目のテストデータの集計を行っていきます。

```
test1 [0] .name [20] , test1 [0] .kokugo, test1 [0] .sansu, test1 [0] .rika,
test1 [0] .shakai
test1 [1] .name [20] , test1 [1] .kokugo, test1 [1] .sansu, test1 [1] .rika,
test1 [1] .shakai
……
test1 [99] .name [20] , test1 [99] .kokugo, test1 [99] .sansu,
test1 [99] .rika, test1 [99] .shakai
```

関数 input_test1 では、構造体変数 test1 にデータを入力していきます。ここでは、**scanf**（スキャンエフ）文を使います。1番目の、

test1[i].name

の入力は、20個の**char**（キャラクタ）型配列に一括して文字を入力するので、変数にはアドレスを調べる「&」記号をつける必要がありません。その他の入力では、通常の**scanf**（スキャンエフ）文と同様、各変数の先頭に「&」をつけます。

gokei_test1 では、4科目それぞれの合計点を計算しています。構造体変数の各メンバーの表記に注意するほかは、通常の足し算と同じです。

heikin_test1 では、4科目の平均点を求めて結果を表示します。注意する点は、gokei_test1 と同じです。

num は、データの件数（何人ぶんか）をかぞえるための変数です。データ入力のときにカウントされ、それが合計や平均の算出に使われます。

PART 6

すぐに役立つ仕事別プログラム

1 ソートプログラム

入力した数字を小さい順に並べかえる

データを大きい順または小さい順に並べかえることを、ソート、ソーティングと呼びます。このプログラムでは、ソーティングのうちでも最も簡単なバブルソートという手法をとっています。バブルソートとは、全データを1つひとつ比べながら大小を判断し、並べかえていく方式です。

関数sortがソートの部分で、nmax個のデータを順に比べていき、小さい順に並んでいなければ入れかえる、という手順になります。なお、データ数はいちおう100個以下としています。

◉ソートプログラム

```
/*  数字を小さい順に並べる  */

#include <stdio.h>

/* 配列xの要素を比べて、小さい順に並べかえるプログラム */

void sort(nmax, x)   ←①
int nmax, x[];
{
    int i, j, work;

    for ( j = 0; j < nmax; j++)              ┐
        for ( i = 0; i <= nmax; i++ )        │
            if ( x[j] < x[i] ) {             ├─②
                work = x[j];                 │
                x[j] = x[i];                 │
                x[i] = work;                 ┘
            }
}

void main()
{
    char n = 0, i, j, nmax;
    int  c, x[100];

/* 正の整数をキーボードから入力し配列xにセットする */
```

```
    printf("32767以下の正の整数を入力してください( END = -1 )¥n");
    do {
        scanf( "%d", &c );
        x[n++] = c;                              ┐── ③
    } while ( c != -1 && n < 100 );

/* 入力した数はnmax、これが２以上のときソートを行う */

    nmax = n - 1;                                ┐── ④
    if ( nmax > 1 ) sort(nmax, x);

/* 個数nmaxが10以上のとき、10個ずつ配列の内容を書き並べる */

    n = 0;
    for ( j = 0; j < nmax / 10; j++ ) {
        for ( i = 0; i < 10; i++ )
            printf ( "%2d  ", x[n++] );          ┐── ⑤
        printf( "¥n" );
    }

/* 残りの個数が10以下のとき、配列の内容を書き並べる */

    for ( i = 0; n < nmax; i++ )                 ┐── ⑥
        printf("%2d  ", x[n++]);
}
```

◉実行結果

```
A>b : d¥sort
32767以下の正の整数を入力してください (END = −1) ←── 表示
12   ┐
35   │
897  ├── 入力
123  │
65   ┘
−1 ←── 入力の終了
12  35  65  123  897 ←── 結果の表示
A>
```

◉フローチャート

開　始
メッセージ
ループ1
キーボードから
整数を入力
入力値→x［n］
n → n+1
c ＝ 1 か n ＞100
ループ1
n − 1 → nmax
nmax
＞1
ソートする
sort(nmax,x)
≦1
0 → n
1

1
ループ2
j＝0,1…, nmax/10
ループ3
i＝0, 1……,9
x［n］を表示
n → n ＋ 1
ループ3
改行する
ループ2
2

2

ループ 4
n＜nmax

x［n］を表示

ループ 4

終　了

sort (nmax, x)

ループ 1
j=0,1…, nmax-1

ループ 2
i=0, 1…, nmax

x［j］＜x［i］

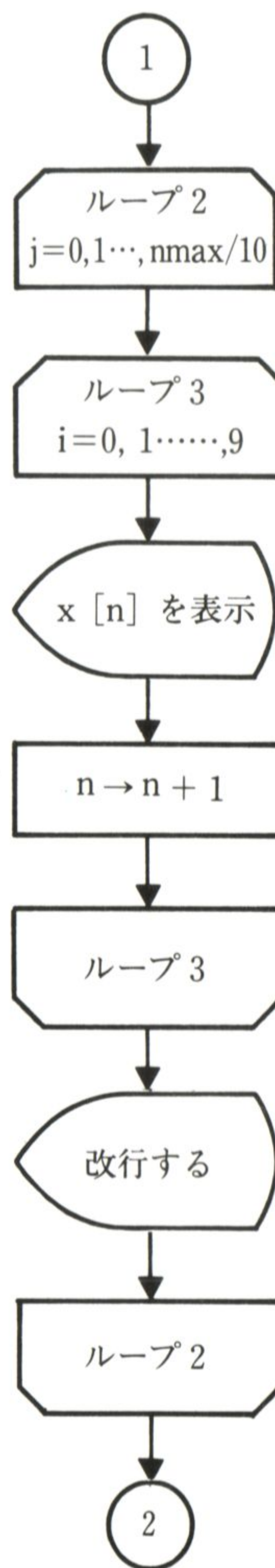
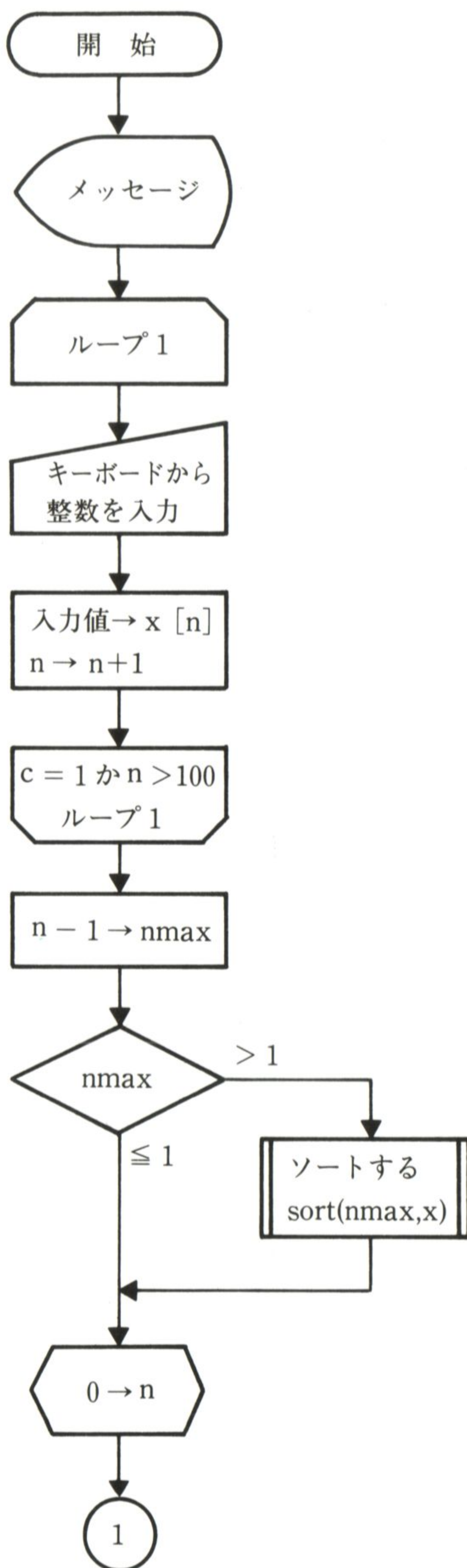
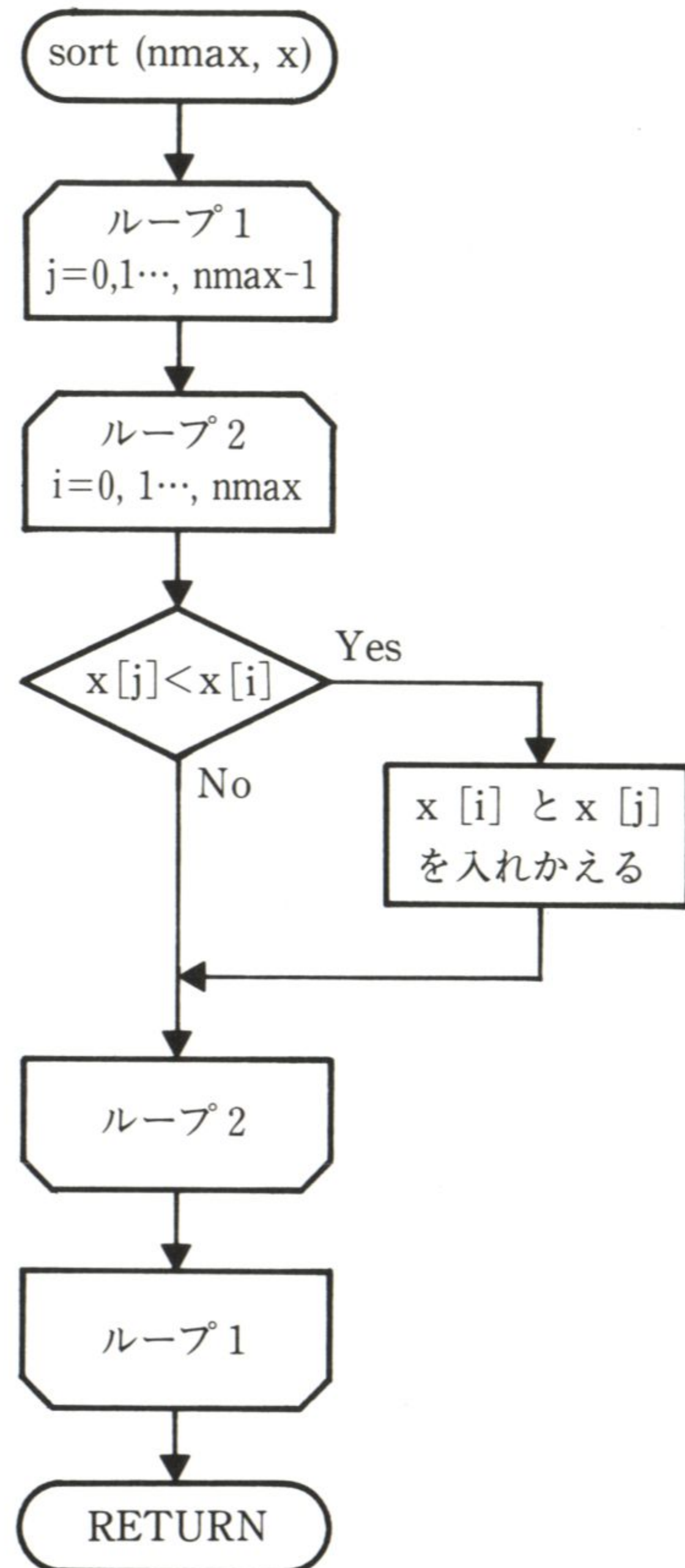
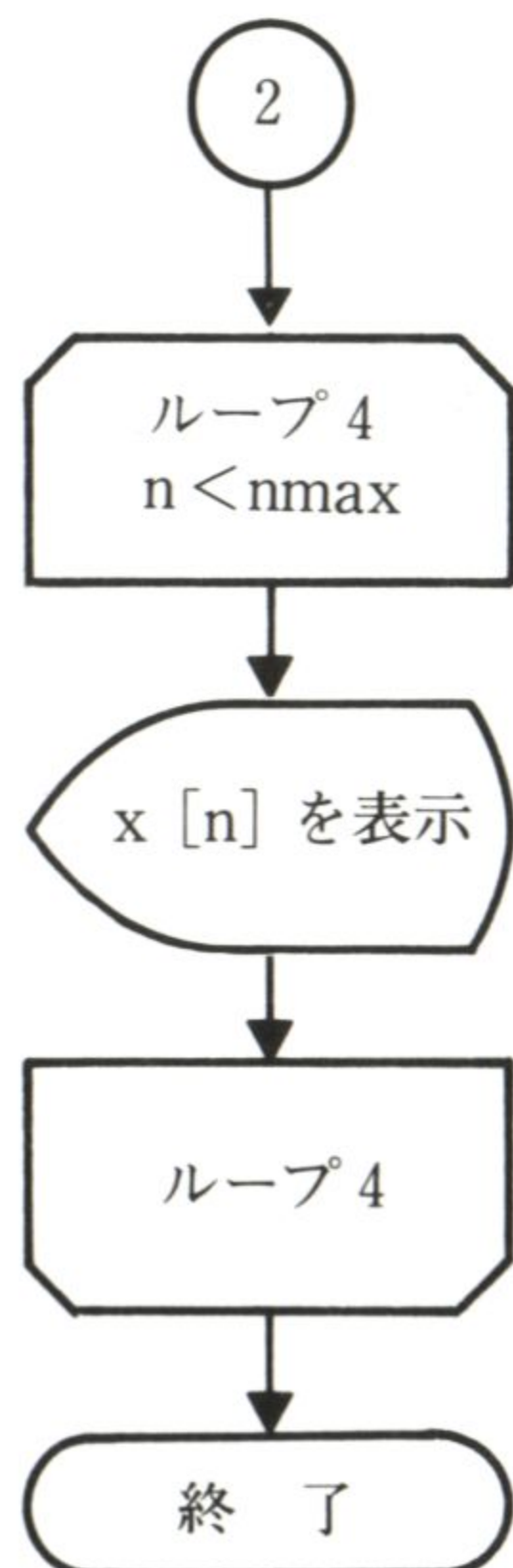
Yes

No

x［i］と x［j］
を入れかえる

ループ 2

ループ 1

RETURN

●プログラムの説明

①関数 sort

ソートを行う部分は、関数 sort です。引数は変数 nmax と配列 x の 2 つですが、配列を引数とすると、関数には配列の先頭アドレスが渡されます。そこで関数のほうでは、

int　x　［100 ］；

と配列の個数を正確に書かずに、プログラムの例のように個数を省略することもできます (x［］)。また、配列の代わりにポインタを定義しておき、配列と同等に扱っても同じ結果になります。たとえば、次のようです。

int　＊x

②バブルソートの本体

この二重の**for**ループが、バブルソートの本体です。全データを1つずつ比べ、小さい順に並べかえていきます。

③データ入力と配列へのセット

データの入力を受けつけて、配列にセットする部分です。**do** ループは最初に必ず1回は実行されるので、まず1個の整数を読み込み、その値が条件を満たしていればもう一度ループを繰り返します。この場合の条件は、読み込んだ値が−1ではなく、しかも配列の個数が100を超えないという複合条件(AND)で、どちらか一方が成立しなくなるとループは終了します。これは、データの個数を100以下に制限していることと、入力の終了を示す場合は「−１」を入力すればよいことを意味します。

④ソート実行の条件

入力した数が１以下の場合、ソートは必要ありません。２以上のときにソートを行います。

⑤結果の表示——1

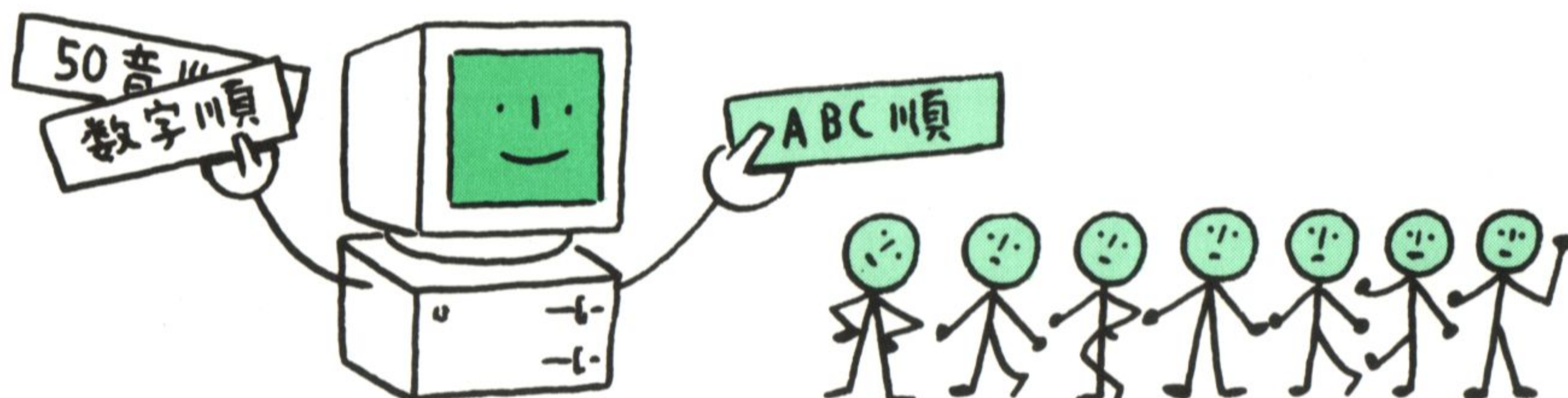
配列 x の内容、つまりソートの結果を表示する部分です。入力した数 nmax が10以上のときは、10個ずつ並べて表示します。

⑥結果の表示——2

残った数が10以下になれば、nmax 個まで x の内容を表示します。

●ワンポイント

ソートには、いろいろな手法があります。データの個数が少ない場合はどの方法をとってもソート時間にそれほど違いはありませんが、データの個数が何万件にもなるときなど、バブルソートは必ずしも効率がよいとはいえません、ほかに、シェルソート、クイックソートなどの方法があります。くわしく知りたい場合は、以下のような文献があります。参照してください。

『応用Ｃ言語』(H.シュルト著／マグロウヒル出版)

2 ダンププログラム

ファイルの内容を画面に表示する

MS-DOS(エムエス・ドス)のテキスト・ファイルの内容を読み込んで、画面に表示するプログラムです。MS-DOSのTYPE(タイプ)コマンドに似ていますが、この場合は、行の左に行番号をつけて表示します。

ファイルのオープン、クローズなどの処理に注意してください。

実行結果では、このプログラムそのものを読み込んで表示しています。

◉ダンププログラム

```
/* テキストファイルの中身を行番号をつけて表示するプログラム */

#include <stdio.h>

void dump(name)
char name[];
{
    FILE *fp;
    char a[256], i = 0;

/* nameというファイルをオープンする */

    fp = fopen( name, "r");  ←――― ①
    if ( fp == NULL ){  ←――― ②
        printf("%s ファイルをオープンできません", name);
        return;
    }

/* ファイルを読み込み、行番号をつけて画面に表示する */
 ┌→ while ( fgets( a, 256, fp ) != NULL )  ←――― ④
 │      printf("%05d   %s", ++i, a );  ←――― ⑤
 ③
/* ファイルをクローズする */
    fclose(fp);  ←――― ⑥
}

void main()
{
    char name[40];
```

```
    printf("ファイル名を入力してください");
    scanf("%s", name);

    dump(name);
}
```

◉実行結果（一部）

```
ファイル名を入力してくださいb：c¥dump1.c ⏎ ←入力
0001  /*テキストファイルの中身を行番号をつけて表示するプログラム*/
0002
0003  # include <stdio.h>
0004
0005  void dump(name)
0006  char name[ ];
0007   {
0008      FILE *fp;
0009      char a[256], i=0;
0010
0011  /*nameというファイルをオープンする*/
0012
0013      fp=fopen( name, "r");
0014      if ( fp==NULL) {
0015              printf("%sファイルをオープンできません",name);
0016              return;
```

●プログラムの説明

①ファイルのオープン

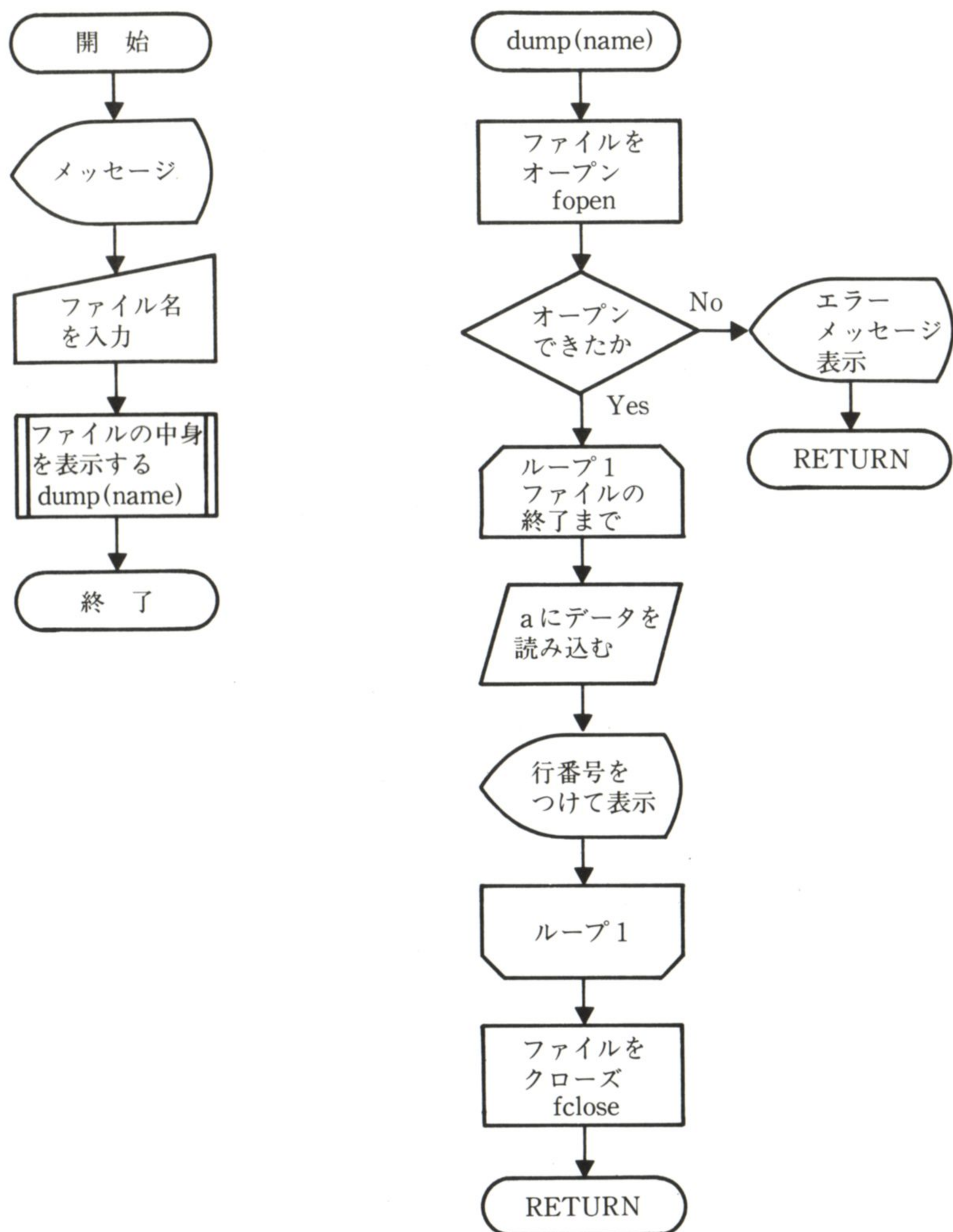
fopen（エフオープン）は、ファイルをオープンする関数です。引数（ひきすう）は次のように書きます。返値（へんち）は、オープンしたファイルのファイルポインタが返されます。「モード」とはそのファイルの性質を表すものです（下記参照）。

fopen（エフオープン）関数の基本書式	fopen（ファイル名，モード）

このプログラムでは、対象となるファイルに関する情報（読み書きしているフ

◉フローチャート

開　始
メッセージ
ファイル名を入力
ファイルの中身を表示する dump(name)
終　了

dump(name)
ファイルをオープン fopen
オープンできたか
No
エラーメッセージ表示
RETURN
Yes
ループ1 ファイルの終了まで
aにデータを読み込む
行番号をつけて表示
ループ1
ファイルをクローズ fclose
RETURN

ァイル上の位置、残りの文字数など）を構造体として扱っています。ファイルポインタとは、このうち、現在読み書きしているファイル上の位置を示すものです。そこで、プログラムの先頭で構造体 FILE のポインタ変数を、次のように定義しておきます。こうしておくと、ポインタ「fp」を使ってファイルをオープンすれば、以降、「fp」を変数のように扱ってファイルを操作することができます。

FILE ＊ fp;

fopen（エフオープン）関数の「ファイル名」には、変数を使うかわりに直接ファイル名を書き込むことができます。その場合は、ファイル名を「”　”」ではさみます。

fopen(”dump.c”,”r”)

モードは、ファイルの内容を読み込んだり、書き出したりする場合に応じて指定します。

”r”	ファイルから読み込む(read)
”w”	ファイルへ書き出す(write)
”a”	ファイルへ追加して書き込む(append)

②ファイルがオープンできないときはNULLすなわち0が返されますから、「(ファイル名) をオープンできません」と表示してmainに戻り、プログラムは終了します（NULLは、ヘッダファイルstdio.hの中で定義されている)。

③次は、ファイルの内容の読み込みです。whileループの条件が真のあいだ、ループがまわります。

④**fgets**（エフゲットエス）は、fpで示されるファイルから指定配列へ、指定のバイト数だけ文字を読み込む関数です。

fgets（エフゲットエス）関数の基本書式	fgets（配列，バイト数，ファイルポインタ）

ファイルの終わりまでくると読み込みができなくなり、NULLが返されるので、ループを抜けます。

⑤読み込んだ内容の表示

この**printf**（プリントエフ）文で、配列に読み込んだ内容を行番号をつけて表示します。

⑥ファイルのクローズ

ファイルをクローズして終わりです。ファイルのクローズを忘れると、正常に動作しないことがありますので注意してください。

●ワンポイント

画面に表示する方法を次のように変えると、MS-DOSのDUMP（ダンプ）コマンドと同じように、16進数のダンププログラムになります。ただし、漢字には対応していませんから、試すときは漢字を使っていないプログラムを対象にしてください。

```
/* ファイルの中身を１６進数で表示するプログラム（漢字非対応） */
```

```
#include <stdio.h>
#include <ctype.h>

void dump(fname)
char fname[20];
{
    FILE *fp;
    unsigned char a[17];
    int kazu,j;
    int i = 0;

/* fnameというファイルをオープンする */
    fp = fopen( fname, "rb");
    if ( fp == NULL ) {
        printf("%s ファイルをオープンできません", fname);
        return;
    }

/* ファイルを読み込み、画面に表示する */
    while( ( kazu = fread( a, 1, 16, fp ) ) != 0 ) {
        printf("%05x : ", i);
        for(j = 0; j < kazu; j++) {
            printf("%02x ", a[j]);
            if( !isprint(a[j]) )  a[j] = '.';
        }
        a[kazu] = '¥0';
        printf("  %s¥n", a);
        i += 16;
    }

/* ファイルをクローズする */
    fclose(fp);
}

void main()
{
    char fname[20];
    printf("ファイル名を入力してください");
    scanf("%s", fname);

    dump(fname);
}
```

◉実行結果

```
A:¥>dump2
ファイル名を入力してくださいc:d¥dump3.c
00000 : 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 69 6f 2e   #include <stdio.
00010 : 68 3e 0d 0a 23 69 6e 63 6c 75 64 65 20 3c 63 74   h>..#include <ct
00020 : 79 70 65 2e 68 3e 0d 0a 0d 0a 76 6f 69 64 20 64   ype.h>....void d
00030 : 75 6d 70 28 66 6e 61 6d 65 29 0d 0a 63 68 61 72   ump(fname)..char
00040 : 20 66 6e 61 6d 65 5b 32 30 5d 3b 0d 0a 7b 0d 0a    fname[20];..{..
00050 : 09 46 49 4c 45 20 2a 66 70 3b 0d 0a 09 75 6e 73   .FILE *fp;...uns
00060 : 69 67 6e 65 64 20 63 68 61 72 20 61 5b 31 37 5d   igned char a[17]
00070 : 3b 0d 0a 09 69 6e 74 20 6b 61 7a 75 2c 6a 3b 0d   ;...int kazu,j;.
00080 : 0a 09 69 6e 74 20 69 20 3d 20 30 3b 0d 0a 0d 0a   ..int i = 0;....
00090 : 09 66 70 20 3d 20 66 6f 70 65 6e 28 20 66 6e 61   .fp = fopen( fna
000a0 : 6d 65 2c 20 22 72 62 22 29 3b 0d 0a 09 69 66 20   me, "rb");...if
000b0 : 28 20 66 70 20 3d 3d 20 4e 55 4c 4c 20 29 20 7b   ( fp == NULL ) {
000c0 : 0d 0a 09 09 70 72 69 6e 74 66 28 22 43 61 6e 6e   ....printf("Cann
000d0 : 6f 74 20 6f 70 65 6e 20 66 69 6c 65 20 25 73 2e   ot open file %s.
```

3 カーソル移動プログラム

カーソルを任意の位置に移動する

プログラムを実行して画面に文字を表示する場合、表示位置の先頭は必ず画面の左端、MS-DOS（エムエス・ドス）のプロンプトの位置になります。画面の文字をいったん全部消し、任意の位置に文字を表示するにはどうしたらよいでしょうか。

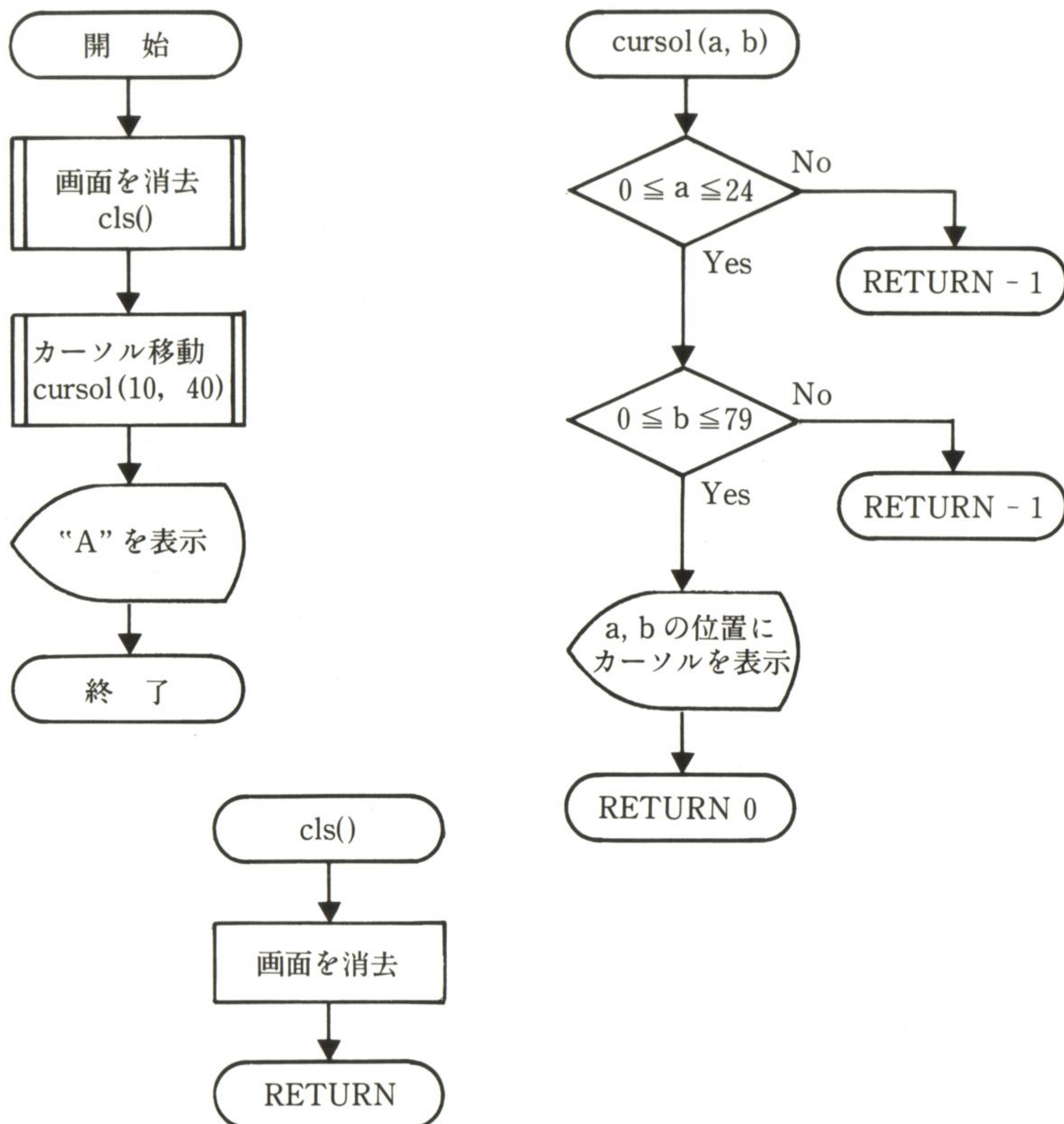
次のプログラムは、画面を消去する関数cls、カーソルを任意の位置に表示する関数cursorを作り、これを利用して画面の中央に文字を表示します。

◎カーソル移動プログラム

```
/* Aという文字を画面の中央に表示するプログラム　*/
#include <stdio.h>

/* 画面全体を消去する */
void cls()
{                                        ①
    printf("¥x1b[2J");
}

/* 指定の行、桁にカーソルを移動する
        行の範囲　1～25まで
              桁の範囲　1～80まで
    範囲外の値はリターンコード-1を返す　　*/

int cursor(a, b)
char a, b;
{
    if ( 1 > a || a > 25 ) return -1;
    if ( 1 > b || b > 80 ) return -1;    ②

    printf("¥x1b[%d;%dH", a, b);  ← ①
    return 0;
}


/* 文字Aを画面の中央に表示します */

void main()
{
```

```
    cls();              /* 画面を消去 */
    cursor(10,40);      /* 画面の中央へカーソル移動 */
    printf("A");      /* Aを表示する */
}
```

◎フローチャート

```
開始
 ↓
画面を消去 cls()
 ↓
カーソル移動 cursol(10, 40)
 ↓
"A"を表示
 ↓
終了

cls()
 ↓
画面を消去
 ↓
RETURN

cursol(a, b)
 ↓
0≦a≦24 —No→ RETURN -1
 ↓Yes
0≦b≦79 —No→ RETURN -1
 ↓Yes
a, bの位置にカーソルを表示
 ↓
RETURN 0
```

●プログラムの説明

①エスケープシーケンス

画面に関する処理を行う関数を持つコンパイラも増えてきましたが、どのコンパイラでも共通というわけにはいきません。ここではMS-DOSのエスケープシーケンス機能を使って、画面の消去、カーソルの表示を行う関数を作ってプログ

ラムにしました。

MS-DOSには、エスケープ・コードとパラメータを使って、画面の制御を行う働きをする機能があります。たとえば、次のようなものです（一部）。

ESC [x ; y H	カーソルをx行y桁の位置に移動
ESC [2J	画面をすべてクリアする
ESC [xA	カーソルをx行上へ移動
ESC [xB	カーソルをx行下へ移動
ESC [xC	カーソルをx桁右へ移動
ESC [xD	カーソルをx桁左へ移動
ESC [a ; … ; zm	a〜zには属性を指示する 0＝既定の属性 1＝ハイライト 2＝バーティカルライン 4＝アンダーライン 5＝ブリンク 7＝リバース

C言語のプログラムでエスケープシーケンスを使用するには、**printf**（プリントエフ）文のカッコ内にエスケープ・コードとパラメータを書きます。エスケープ・コードは16進数では1bですので、プログラム中に入れるときは先頭に「¥x」をつけて「¥x1b」とします。パラメータは、たとえば次のようです。

```
printf("¥x1b [2J");               ……画面を消去する
printf("¥x1b [10;10H");           ……10行10桁にカーソルを表示する
printf("¥x1b [%d;%dH", x, y);     ……パラメータが入るところを変数にして、x行
                                       y桁にカーソルを表示する
```

②入力ミスの処理

テキスト画面の座標は25行、80桁が標準ですから、関数を呼ぶときに、これ以上の行数や桁数を指定したときは、−1を返して終了します。

●ワンポイント

この例の場合、行数と桁数は固定値（行10、桁40）で使っているので変数a、bは不要ですが、汎用性を持たせるために作りました。任意の値を入力できるようにしておけば、その位置に文字を表示させることができます。

4 素数検出プログラム

任意の数までの素数を求める

素数というのは、1、2、3……の自然数のうち、自分自身と1以外では割りきれない数のことです。10までの素数には2、3、5、7があり、このくらいなら少し考えれば見つけることができるでしょう。しかし、数字が大きくなると、素数を判定するのは、なかなかたいへんな作業になります。

素数判定のような作業は、コンピュータの最も得意とするところです。ここでは、1万個の配列（a［10000］）を**static**（スタティック）（静的）変数としてあらかじめ確保し、そこに求めた変数を入れていきます。**static**（スタティック）変数は、一度メモリを割り当てておくと、プログラムの終了までその位置に存在する変数で、**auto**（オート）変数ではメモリオーバーになるような大きな配列を定義するときなどに用いられます（ふろく：187ページ参照）。

◉素数検出プログラム

```
/* 素数を求めるプログラム */

main() {
    static int a[10000];                          ①
    int l = 2, k = 1, i, nmax, flg;

    printf("¥nいくつまでの素数を求めますか？ ");      ②
    scanf("%d", &nmax);

    printf("¥n*******************************************¥n");   ③
    printf("では自然数 %d までの素数を求めましょう¥n", nmax);
    printf("*******************************************¥n");

    printf("a[  0] =   1       ");   ←④

    while ( l <= nmax ) {   ←⑤
        flg = 0;
        for ( i = 1; i <k; i++ )
            if ( l % a[i] == 0 ) flg = 1;   ←⑥

        if ( flg == 0 ) {
            a[k] = l;
            printf("a[%3d] = %3d       ", k, a[k]);   ←⑦
```

```
            if ( (k+1)%4 == 0 ) printf("¥n");
            k++;
        }
        l++;
    }

    printf("¥n*********************************************¥n");  ─┐
    printf(" %d までの素数は %d 個でした¥n", nmax, k);              ⑧
    printf("*********************************************¥n");   ─┘

}
```

◉実行結果

```
いくつまでの素数を求めますか? 50↵ ←──入力

*********************************************
では自然数 50 までの素数を求めましょう
*********************************************
a[  0] =   1      a[  1] =   2      a[  2] =   3      a[  3] =   5
a[  4] =   7      a[  5] =  11      a[  6] =  13      a[  7] =  17
a[  8] =  19      a[  9] =  23      a[ 10] =  29      a[ 11] =  31
a[ 12] =  37      a[ 13] =  41      a[ 14] =  43      a[ 15] =  47

*********************************************
 50 までの素数は 16 個でした
*********************************************
```

●プログラムの説明

①使用する変数

このプログラムで使用する変数は、以下のとおりです。

a [10000]	素数を入れる配列。最大1万個まで可能
l	自然数のカウンタ
k	素数の番号（数）
i	for ループのカウンタ
nmax	素数を求める自然数の最大値（入力する）

◉フローチャート

開　始

入力用のメッセージ

自然数を入力

はじめのメッセージ

ループ1
l≦nmax

flg=0

ループ2
i=1, 2…, k

lがa[i]で割り切れる

Yes

flg=1

No

ループ2

1

1

flg : 0

=

a [k]=l

素数a [k]を表示

k+1が4の倍数

Yes

改行

No

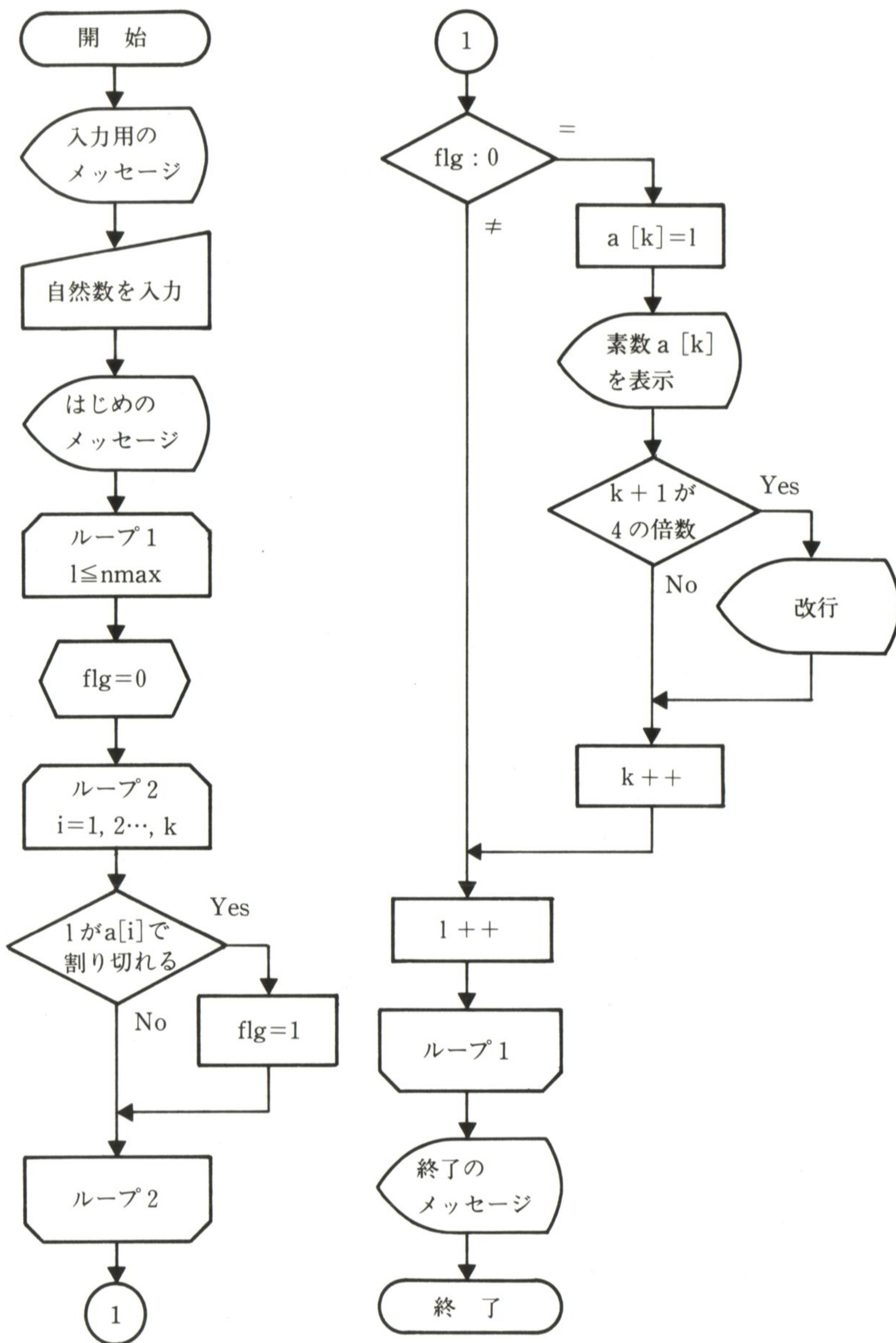
≠

k++

l++

ループ1

終了のメッセージ

終　了

flg　　　　　　lがほかの素数で割りきれるとき、1にしておく

②最大値の入力

メッセージを表示し、素数を求める自然数の最大値の入力を待ちます。

③スタートの表示

次のメッセージを飾りつきで表示します。

　では自然数 ○ までの素数を求めましょう

④「1」の表示

厳密にいうと1は素数ではありませんが、a[0] には1を入れておき最初に表示しましょう。

⑤素数判定のループ

lは判定したい自然数で、それがnmax以下のあいだはループを繰り返します。kは、すでに見いだされた素数の数です。

⑥lが素数でないときの処理

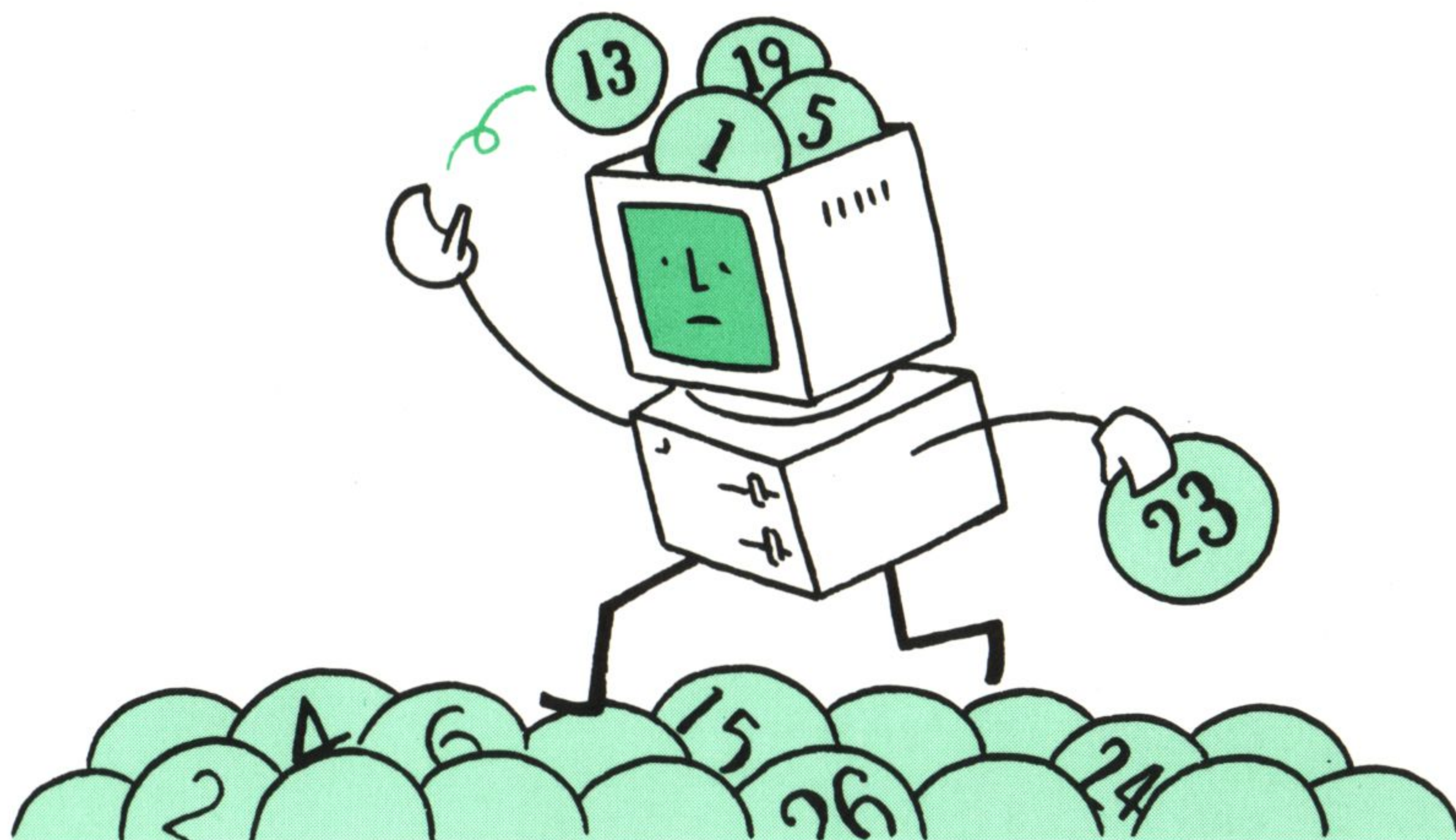
判定したい自然数lを、すでに見いだされた素数 a[i] で割り、余りが0のときはflgに1を代入し、ループから抜けます。

⑦素数のセットと表示

flgが0のとき、lは素数ですから、それを配列 a[k] にセットします。そして、a[k] つまり素数を、画面の1行に4個ずつ表示します。

⑧結果のメッセージ

次のメッセージを飾りつきで表示して、プログラムを終了します。

　○ までの素数は ○ 個でした

5 グラフィックプログラム

ブタの絵を描く

グラフィック関数は、C言語では標準化されていません。コンパイラによってはグラフィック関数を備えたものもありますが、関数名や使用方法がまちまちなので、グラフィック関数を使ってプログラミングを行う場合、コンパイラによってその部分を書きかえる必要があります。

ところが、MS-DOS(エムエス・ドス) Ver3.3以上のバージョンには「GRAPH. SYS」というグラフィックドライバが搭載されたため、これを使ってグラフィック関数を作ることが可能になりました。CONFIG. SYS ファイルの中に、

DEVICE＝GRAPH. SYS

を指定して、「GRAPH. SYS」ドライバを起動し、MS-DOS のファンクションコールという方法でプログラミングを行います。ここに掲げた例題は、このやりかたで直線と四角形を描く関数 Line、楕円を描く関数 Ellips、三角形を描く関数 Triangle を作り、ブタの絵を描いてみました。

プログラムが大きくなるので、いろいろな宣言はヘッダファイル graphic. h にまとめました。このファイルは、コンパイラ include ファイルのあるディレクトリに入れておきます。プログラム本体では「# include ＜graphic. h＞」の1行を書いておくと、コンパイル時点でヘッダファイル graphic. h に置きかえられます。

MS-DOS のファンクションコールのやりかたは、初心者にはかなりむずかしいものです。実際にグラフィックスを使ってみようという場合は、C言語のコンパイラごとに備えつけられているグラフィックライブラリを使ったほうが無難でしょう。ここではファンクションコールのやりかたまでは説明しませんが、各関数を利用して引数(ひきすう)の値を変えて使えば、いろいろな絵を描くことができます。

●関数の説明

screen(x)……グラフィック画面のモードを次のように指定する

x＝0：640×200、白黒モード

x＝1：640×200、8色カラーモード

x＝2：640×400、高分解白黒モード

x＝3：640×400、8色高分解カラーモード

cls(x)……画面をクリアする

x＝1：テキスト画面のみクリア

x＝2：グラフィック画面のみクリア

x＝3：テキスト画面とグラフィック画面をクリア

Line (x1, y1, x2, y2, Fl, LineClr, Fl2, PaintClr)

……直線または四角形を描く

x1, y1　　：始点の座標（x1, y1）

x2, y2　　：終点の座標（x2, y2）

Fl　　　　：直線の場合は0、四角形の場合は1を指定する

LineClr　：線分の色を指定する

Fl2　　　：塗りつぶし＝1、塗りつぶさない＝0（四角形の場合のみ）

PaintClr　：塗りつぶしの色

Ellips (xc, yc, rl, rs, Fl, xs, ys, xe, ye, LineClr, Fl2, PaintClr)

……楕円を描く

xc, yc　　：中心の座標（xc, yc）

rl, rs　　：楕円の長径、短径

Fl　　　　：0＝楕円　1＝楕円弧　2＝扇形

xs, ys　　：開始点の座標（xs, ys）（楕円弧と扇形の場合）

xe, ye　　：終了点の座標（xe, ye）（　　〃　　）

LineClr　：線分の色を指定する

Fl2　　　：塗りつぶし＝1、塗りつぶさない＝0

PaintClr　：塗りつぶしの色

Triangle (x1, y1, x2, y2, x3, y3, LineClr, Fl, PaintClr)……三角形を描く

x1, y1, x2, y2, x3, y3：三角形の3点の座標

(x1, y1) － (x2, y2) － (x3, y3)

LineClr　：線分の色を指定する

Fl　　　　：塗りつぶし＝1、塗りつぶさない＝0

PaintClr　：塗りつぶしの色

close( )……グラフィックスを消して初期状態に戻す

色の番号――0：黒　1：青　2：緑　3：水　4：赤　5：紫　6：黄

7：白

```
/* ヘッダーファイル  graphic.h        */
#include <stdio.h>
#include <dos.h>
#include <math.h>

union REGS inregs, outregs;

struct FuncEntryTags {
    int (far pascal *FC[41])(unsigned long);
    } far *GL;          /* エントリーテーブルの先頭アドレスを格納する */

union GrpDataTag {
    unsigned int work[1024];                /* 作業領域 */
    struct LineDATA {                   /* 線または四角形の属性を指定する */
        unsigned long Reserved;          /* 予約パラメータ=0 */
        unsigned int Rop;               /* ラスタオペレーション番号 */
        unsigned int Xs;                /* 始点のx座標 */
        unsigned int Ys;                /* 始点のy座標 */
        unsigned int Xe;                /* 終点のx座標 */
        unsigned int Ye;                /* 終点のy座標 */
        unsigned char Flg;              /* 描画フラグ */
        unsigned char FlgDum;           /* 描画フラグ */
        unsigned long Pno;              /* 線の色を示すパレット番号 */
        unsigned int Wide;              /* 線の幅 */
        unsigned int LinePtL;           /* 線種パターン長 */
        unsigned int LinePt;            /* 線種パターン */
        unsigned int LinePtE;           /* 拡張線種パターン */
        unsigned char PtFlg;            /* 塗りつぶしフラグ */
        unsigned char PtFlgDum;         /* 塗りつぶしフラグ */
        unsigned long PaintNo;          /* 塗りつぶし色のパレット番号 */
        unsigned int PaintPt;           /* 塗りつぶしのタイルパターン長 */
        unsigned long PatnAdr;          /* タイルパターン格納域アドレス */
    } Ldata;

    struct EllipsDATA {                 /* 楕円の属性を指定する */
        unsigned long Reserved;          /* 予約パラメータ=0 */
        unsigned int Rop;               /* ラスタオペレーション番号 */
        unsigned int Xc;                /* 中心点x座標 */
        unsigned int Yc;                /* 中心点y座標 */
        unsigned int Xr;                /* x方向半径 */
        unsigned int Yr;                /* y方向半径 */
        unsigned int Xs;                /* 開始点x座標 */
        unsigned int Ys;                /* 開始点y座標 */
```

```
    unsigned int Xe;               /* 終了点y座標 */
    unsigned int Ye;               /* 終了点y座標 */
    unsigned char Flg1;            /* 描画フラグ */
    unsigned char Flg1Dum;         /* 描画フラグ */
    unsigned long Pno;             /* 線の色のパレット番号 */
    unsigned char Flg2;            /* 形状フラグ */
    unsigned char Flg2Dum;         /* 形状フラグ */
    unsigned int LinePtL;          /* 線種パターン長 */
    unsigned int LinePt;           /* 線種パターン */
    unsigned int LinePtE;          /* 拡張線種パターン */
    unsigned char Flg3;            /* 塗りつぶしフラグ */
    unsigned char Flg3Dum;         /* 塗りつぶしフラグ */
    unsigned long PaintNo;         /* 塗りつぶし色のパレット番号 */
    unsigned int PaintPt;          /* 塗りつぶしのタイルパターン長 */
    unsigned long PatnAdr;         /* タイルパターン格納域アドレス */
} Edata;

struct TriangDATA {                /* 三角形の属性を指定する */
    unsigned long Reserved;        /* 予約パラメータ=0 */
    unsigned int Rop;              /* ラスタオペレーション番号 */
    unsigned int X1;               /* 第1x座標 */
    unsigned int Y1;               /* 第1y座標 */
    unsigned int X2;               /* 第2x座標 */
    unsigned int Y2;               /* 第2y座標 */
    unsigned int X3;               /* 第3x座標 */
    unsigned int Y3;               /* 第3y座標 */
    unsigned char Flg1;            /* 描画フラグ */
    unsigned char Flg1Dum;         /* 描画フラグ */
    unsigned long Pno;             /* 線の色のパレット番号 */
    unsigned int Wide;             /* 線幅 */
    unsigned int LinePtL;          /* 線種パターン長 */
    unsigned int LinePt;           /* 線種パターン */
    unsigned int LinePtE;          /* 拡張線種パターン */
    unsigned char PtFlg;           /* 塗りつぶしフラグ */
    unsigned char PtFlgDum;        /* 塗りつぶしフラグ */
    unsigned long PaintNo;         /* 塗りつぶし色のパレット番号 */
    unsigned int PaintPt;          /* 塗りつぶしのタイルパターン長 */
    unsigned int PatnAdr;          /* タイルパターン格納域アドレス */
    } Tdata;

struct DispSwitch {
    unsigned long Dum1;
    unsigned int Dps;
} Dsdata;
```

```
    struct DispMode {
        unsigned long Vram;
        unsigned int Dum2;
    } Dmdata;
} Grdata;

/* グラフィックプログラム  graphic.c */
#include <graphic.h>

/* ファンクションコール */
GraphFunc(x)
int x;
{
    (*(GL->FC[x]))((unsigned long)(int far *)&Grdata);
}


/* スクリーンの初期状態を決める */
screen(x)
int x;
{
    GraphFunc(0);                  /* グラフィックの初期設定を行う */
    Grdata.Dmdata.Vram = 0;
    switch (x) {
    case 0: Grdata.Dmdata.Dum2 = 0x0000;break;
    case 1: Grdata.Dmdata.Dum2 = 0x0001;break;
    case 2: Grdata.Dmdata.Dum2 = 0x0100;break;
    case 3: Grdata.Dmdata.Dum2 = 0x0101;break;
    }
    GraphFunc(3);                  /* スクリーンのモードを決める */
    Grdata.Dsdata.Dps = 0x101;
    GraphFunc(11);                 /* 表示スイッチの設定 */
}

/* 画面をクリア  x = 0 :テキスト画面 1:グラフィック画面 2:両方 */
cls(x)
int x;
{
    if ( x == 1 || x == 3 ) printf("¥x1b[2J");/* テキスト画面のクリア */
    if ( x == 2 || x == 3 ) {
    Grdata.Edata.Reserved = 0;
    GraphFunc(14);                    /* グラフィック画面のクリア */
    }
}

/* グラフィックを終了 */
GrpEnd()
```

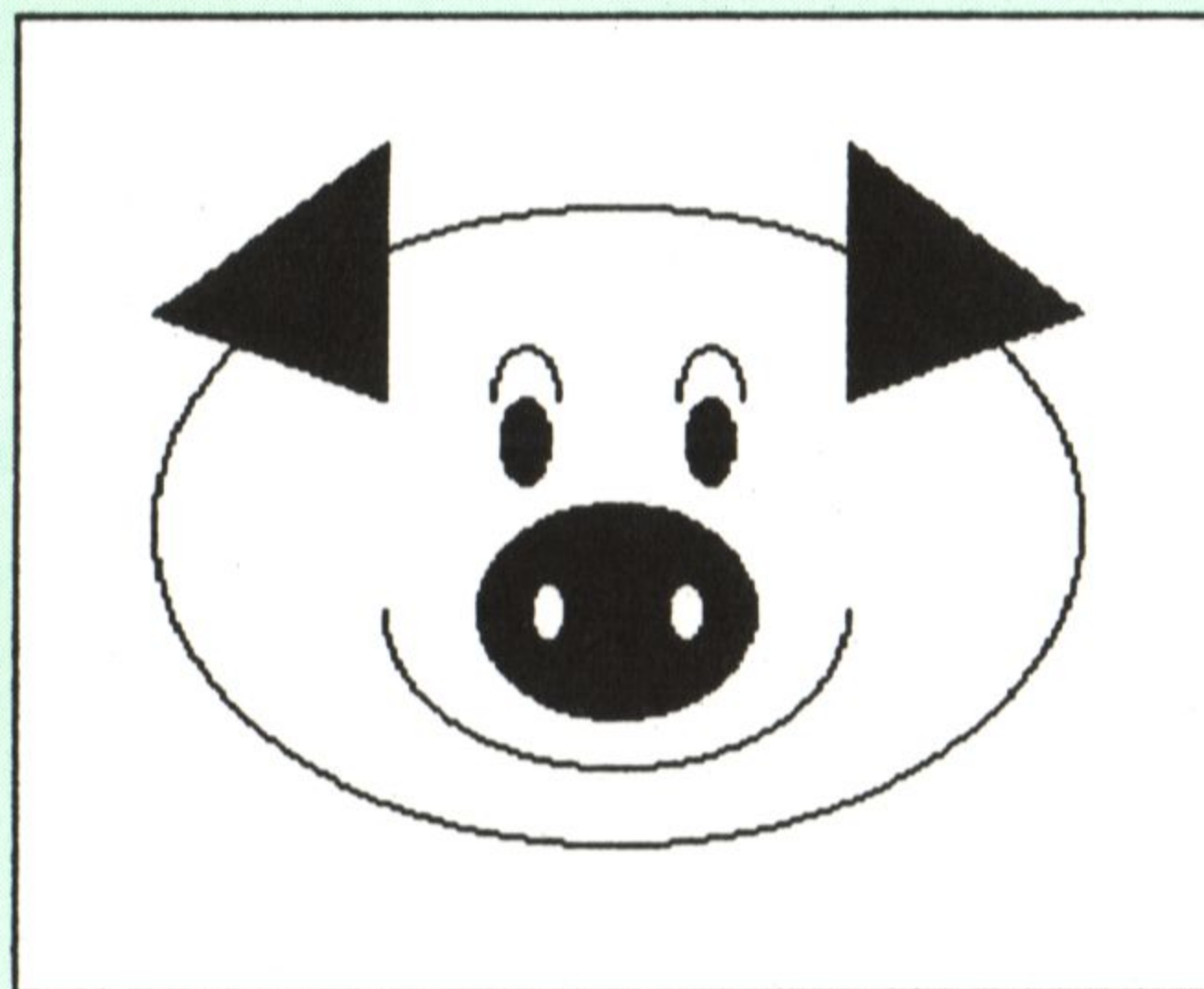
```
{
    GraphFunc(1);                          /* グラフィックの終了 */
}

/* 線分と四角形を描く */
Line(x1, y1, x2, y2, Fl, LineClr, Fl2, PaintClr )
unsigned int x1, y1, x2, y2;
int Fl, Fl2;      /* Fl = 0:直線    Fl = 1:四角形  Fl2 = 1 塗りつぶし */
long LineClr, PaintClr;
{
    Grdata.Ldata.Reserved = 0;
    Grdata.Ldata.Rop = 0;
    Grdata.Ldata.Flg = 0;
    Grdata.Ldata.Pno = LineClr;
    if ( Fl2 == 1 ) Grdata.Ldata.PtFlg = 0x01;
    else Grdata.Ldata.PtFlg = 0x00;
    Grdata.Ldata.PaintNo = PaintClr;
    Grdata.Ldata.Xs = x1;
    Grdata.Ldata.Ys = y1;
    Grdata.Ldata.Xe = x2;
    Grdata.Ldata.Ye = y2;

    if ( Fl == 1 ) GraphFunc(16);                    /* 線分を描く */
    else GraphFunc(18);                              /* 四角形を描く */
}

/* 楕円を描く */
Ellips(xc, yc, rl, rs, Fl, xs, ys, xe, ye, LineClr, Fl2, PaintClr)
unsigned int xc, yc, rl, rs, xs, ys, xe, ye;
int Fl, Fl2;
unsigned long LineClr, PaintClr;
{
    Grdata.Edata.Reserved = 0;
    Grdata.Edata.Rop = 0;
    Grdata.Edata.Xc = xc;
    Grdata.Edata.Yc = yc;
    Grdata.Edata.Xr = rl;
    Grdata.Edata.Yr = rs;
    Grdata.Edata.Flg1 = 0x00;
    Grdata.Edata.Pno  =  LineClr;
    switch (Fl) {                       /* 楕円、楕円弧、扇形を選ぶ */
        case 0: Grdata.Edata.Flg2 = 0x00;break;
        case 1: Grdata.Edata.Flg2 = 0x05;break;
        case 2: Grdata.Edata.Flg2 = 0x0f;
    }
```

```
    Grdata.Edata.Xs = xs;
    Grdata.Edata.Ys = ys;
    Grdata.Edata.Xe = xe;
    Grdata.Edata.Ye = ye;
    if ( Fl2 == 1 ) Grdata.Edata.Flg3 = 0x01;
    else Grdata.Edata.Flg3 = 0x00;
    Grdata.Edata.PaintNo =  PaintClr;
    GraphFunc(21);
}

/* 三角形を描く */
Triangle(x1, y1, x2, y2, x3, y3, LineClr, Fl, PaintClr)
unsigned int x1, y1, x2, y2, x3, y3;
unsigned long LineClr, PaintClr;
int Fl;
{
    Grdata.Tdata.Reserved = 0;
    Grdata.Tdata.Rop = 0;
    Grdata.Tdata.X1 = x1;
    Grdata.Tdata.Y1 = y1;
    Grdata.Tdata.X2 = x2;
    Grdata.Tdata.Y2 = y2;
    Grdata.Tdata.X3 = x3;
    Grdata.Tdata.Y3 = y3;
    Grdata.Tdata.Flg1 = 0x00;
    Grdata.Tdata.Pno = LineClr;
    if ( Fl == 1 ) Grdata.Tdata.PtFlg = 0x01;
    else Grdata.Tdata.PtFlg = 0x00;
    Grdata.Tdata.PaintNo = PaintClr;
    GraphFunc(17);              /* 三角形を描く */
}


main()
{
    long c0 = 0, c1 = 1, c2 = 2, c3 = 3, c4 = 4, c5 = 5, c6 = 6, c7 = 7;

    int c;
    inregs.x.ax = 0;
    inregs.x.bx = (int)&GL;
    int86(0xcd, &inregs, &outregs);

    screen(3);
    cls(3);
    printf("Graphic program start:¥n");
```

```
/*  Line(x1, y1, x2, y2, F1, LineClr, F12, PaintClr ) */
    Line( 120,  60, 380, 290, 0, c7, 0, 0);


/*  Ellips( xc, yc, r1, rs, F1,  xs, ys, xe, ye,LineClr,F12,PaintClr) */
    Ellips(250, 180, 100, 75, 0,   0,  0,  0,   0, c6, 0, 0);
    Ellips(250, 200,  30, 25, 0,   0,  0,  0,   0, c4, 1, c4);
    Ellips(235, 200,   3, 6, 0,   0,   0,  0,  0, c0, 1, c0);
    Ellips(265, 200,   3, 6, 0,   0,   0,  0,  0, c0, 1, c0);

    Ellips(250, 200,  50, 37, 1, 200, 200,300,200, c6, 0,  0);
    Ellips(230, 160,   5, 10, 0,   0,   0,  0,   0, c7, 1, c7);
    Ellips(270, 160,   5, 10, 0,   0,   0,  0,   0, c7, 1, c7);
    Ellips(230, 150,   7, 12, 1, 237, 150,213,150, c2, 0,  0);
    Ellips(270, 150,   7, 12, 1, 277, 150,263,150, c2, 0,  0);

/*  Triangle(x1, y1, x2, y2, x3, y3, LineClr, F1, PaintClr) */
    Triangle(150,130,200,150,200, 90, c7,1,c7);
    Triangle(300,150,300, 90,350,130, c7,1,c7);

    c = getch();
    GrpEnd();
    printf("Graphic program end:¥n");
}
```

◉実行結果

```
Graphic program start:
```

ふろく 1

変数について

本文では変数の4つの型（char（キャラクタ）型、int（インテジャ）型、float（フロート）型、double（ダブル）型）について説明をしましたが、このほかにもいくつかの変数の型があります。

long型とunsigned型

まず、整数型でint（インテジャ）の2倍の大きさを持つlong（ロング）型。これはint（インテジャ）型では表現できない整数を入れるのに使います。int（インテジャ）型では−32,768〜＋32,767までの整数しか入りません。

```
#include <stdio.h>
main()
{
    int a,b;

    a = 5000;
    b = 50000000;

    printf("a = %d, b = %ld", a, b);
}
```

この例では、bは50000000になるはずですが、実際にこのプログラムを動かしてみると、bはおかしな値が表示されます。これは、bがint（インテジャ）型だったために、掛け算の結果を入れようとして桁あふれを起こしたのです。そこで、

```
int a;
long b;
```

のように修正すると、bには正しい値が入ります。long（ロング）型の変数の大きさは4バイトなので−2,147,483,648〜＋2,147,438,647までの整数を入れることができます。

int（インテジャ）、long（ロング）は、正しくは次のようないいかたの略ですが、通常int（インテジャ）型、long（ロング）型と呼んでいます。

short int → int
long int → long

char(キャラクタ)や**int**(インテジャ)には正および負の整数を入れることができますが、特に正の整数だけを使いたい場合は**unsigned**(アンサインド)をつけて、

```
unsigned char a, b;
unsigned int c, d;
```

のように変数宣言することができます。この場合、変数a、bには0～255、変数c、dには0～65,535までの値を入れることができます。

auto変数とstatic変数

通常、プログラムの中で宣言されるのは**auto**(オート)（自動）変数で、これはスタックと呼ばれる領域に記憶される変数です(89ページ参照)。関数の内部で定義される変数はすべて**auto**(オート)変数ですので、関数を2回呼んだ場合、値は保存されず必ず初期値に戻ります。

2回目にも値を残しておきたいようなときは、次のように**static**(スタティック)変数というものを宣言します。

```
/* 呼ばれた回数をカウントする関数 */

count()
{
    static int n;

    n = n + 1;
}
```

static(スタティック)変数はメモリ上の別の場所に確保されるので、いったん値をセットすると、ずっと保存されています。これが静的変数といわれるゆえんです。もっとも、これは定義した関数の中だけで機能するローカル変数です。

int(インテジャ)だけでなく、**char**(キャラクタ)、**float**(フロート)、**double**(ダブル)などの先頭に**static**(スタティック)をつけて宣言すると**static**(スタティック)変数になります。

static(スタティック)変数は、関数の中で大きな配列を宣言する場合や、配列に初期値をセットする場合などに使われます。

配列に初期値を入れる

変数の場合は、

```
int x = 0;
```

のように、変数宣言と同じ行で初期値を入れることもできましたが、配列の場合はどうでしょうか。前にも触れたように、配列の場合は、通常の位置に宣言する**auto**(オート)変数に続けて初期値を入れることはできません。そこで**static**(スタティック)変数を使うと、これと同じようなことができます。

```
main()
{

    static int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

……
}
```

このように「{　}」の中に「,」で区切って値を入れていくと、順にa[0]、a[1]、a[2]……にセットされます。初期値を入れない場合は、あらかじめ0がセットされます。

配列が**char**(キャラクタ)型の場合は、

```
main()
{

    static char a[10] = { 'I', 'm', 'a', 'g', 'i', 'n', 'e', '¥0' };

……
}
```

のように1つずつ区切って文字を入れてもよいのですが、次のように書くこともできます。この場合も上と同じ内容がa[0]から順にセットされ、最後のa[7]には自動的に「'¥0'」の1文字が入ります。

```
main()
{

    static char a[10] = "Imagine";

……
}
```

右辺で初期値を代入する場合は、配列の大きさは指定せず、

```
static char a[] = "Imagine";
```

のように書いておくと、代入するデータに合わせて、この場合は8個の配列が用意されます。

変数の通用範囲（ローカル変数とグローバル変数）

これまでに出てきた**auto**（オート）変数や**static**（スタティック）変数は、main や関数の「{　}」の中で定義されていました。これらは局所的に使われる変数で、ローカル変数と呼ばれています。main の「{　}」の中で宣言された場合はその中だけで、関数の「{　}」の中で宣言された場合はその中だけで有効な変数です。また89ページのコラムで説明したように、さらに小さいブロックの中だけで使われることもあります。変数は、それぞれ「{　}」の中の宣言した場所以降で有効です。

これに対して、もっと広い範囲に通用する変数があり、グローバル変数と呼ばれます。グローバル変数は通常、プログラムの先頭で定義されます。

```
#include <stdio.h>
int a, b;   ←―― グローバル変数
doubl()
{
    a = 2 * a;
    b = 2 * b;
}

main()
{
    a = 12;
    b = 24;
    doubl();
    printf("a = %d, b = %d", a, b);
}
```

a，b は main の中でも関数 doubl の中でも同様に扱われるので、実行すると、

```
a = 24, b = 48
```

が表示されます。155ページの構造体変数 test1 も、グローバル変数の一例です。

プログラムの先頭でなく、プログラムの中間で次のように定義された場合は、それ以降のプログラムに関して（ここでは plus10、main の中で）、使用することができます。

```
#include <stdio.h>
double(a, b)
{
    a = 2 * a;
    b = 2 * b;
}

int a, b
plus10()
{
    a = a + 10;
    b = b + 10;
}
main()
{
    a = 12;
    b = 24;
    doubl(a, b);
    printf("a = %d, b = %d", a, b);
    plus10();
    printf("a = %d, b = %d", a, b);
}
```

ローカル変数、グローバル変数という呼び分けかたは、変数の通用範囲による相対的な分類です。カーニハン＆リッチーによる『プログラミング言語C』には、グローバル変数は、むしろ外部変数として厳密に定義されています（次項参照）。そして**static**（スタティック）変数は、定義される場所によってローカルである場合と、グローバルである場合とがあります。

extern 変数とファイルの分割

グローバル変数は関数および main の外部で定義されるので、正しくは外部変数（**extern**（エクスターナル）変数）と呼ばれるべきものです。定義のしかたもは

```
int a, b
```

と書かれます。外部変数は定義された場所以降、プログラムの終わりまで有効で

す。外部とはつまり、関数の外で宣言されているという意味です。

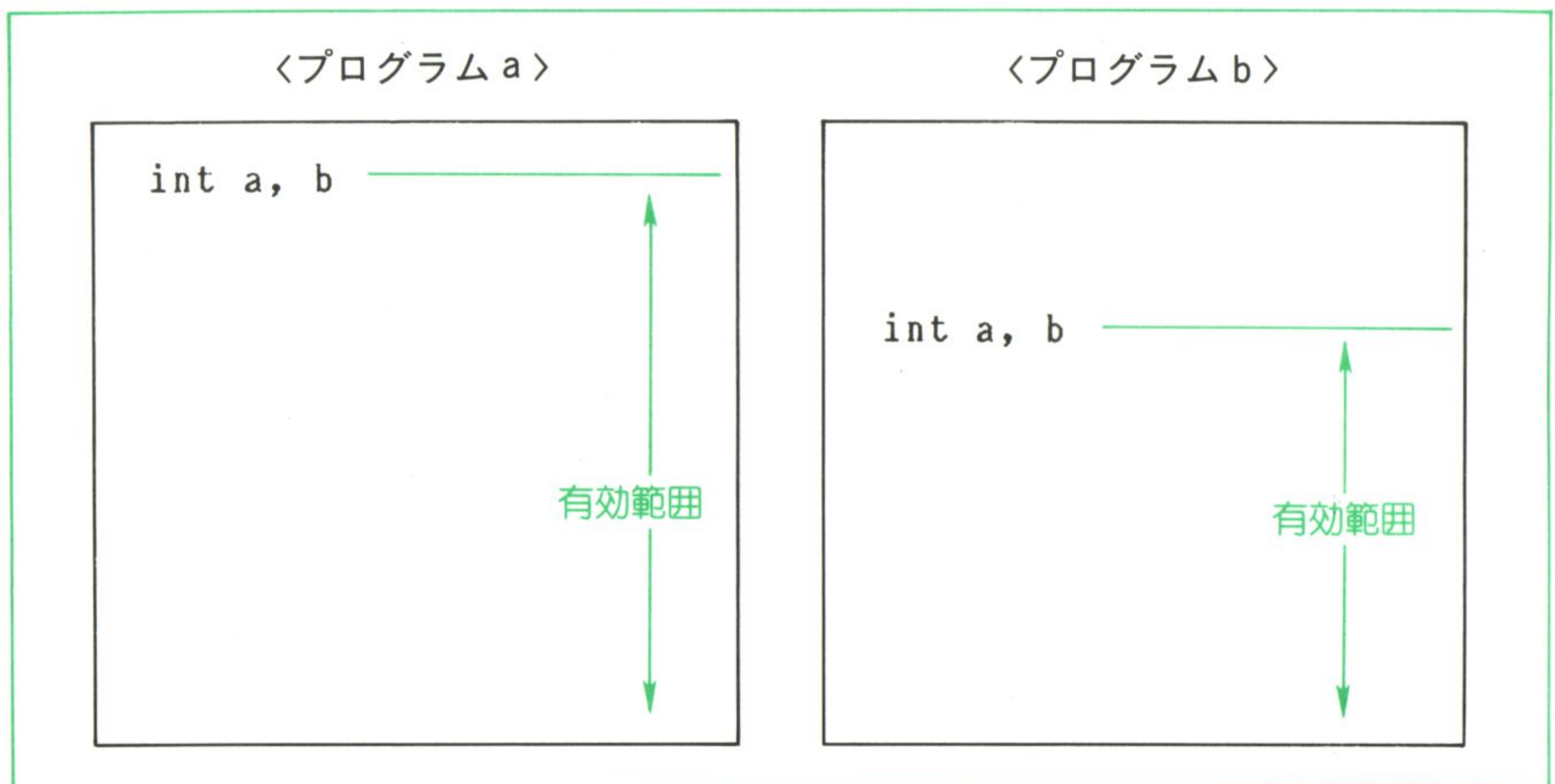
〈プログラムａ〉

```
int a, b
```

有効範囲

〈プログラムｂ〉

```
int a, b
```

有効範囲

ところが、プログラムが大きくなると、２つ以上のファイルに分けることがあります。複数のファイルに分ける場合、別ファイルから参照する部分には、必ず**extern**（エクスターナル）宣言をしてから使用しなければなりません。

〈プログラムｃ〉

ここで宣言する

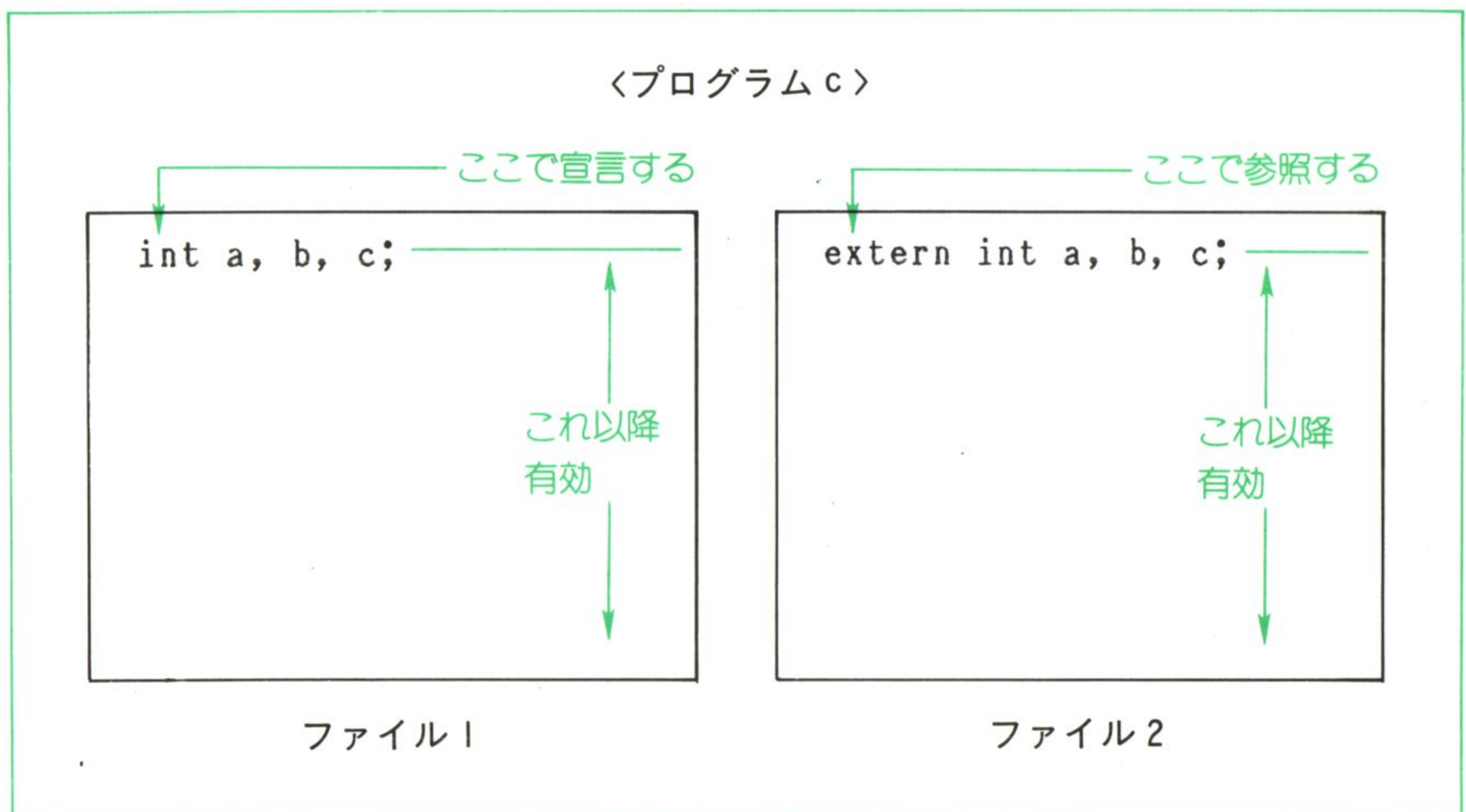
```
int a, b, c;
```

これ以降有効

ファイル１

ここで参照する

```
extern int a, b, c;
```

これ以降有効

ファイル２

この例では、ファイル１で外部変数を定義し、ファイル２で同じ変数を使用するために**extern**（エクスターナル）宣言をしています。

ファイルの数が多くなると、各ファイルで**extern**（エクスターナル）宣言をするのは複雑ですの

で、〇〇〇〇. hという名前をつけたヘッダファイルにまとめて

　　#include　"〇〇〇〇. h"

または

　　#include 〈〇〇〇〇. h〉

という**include**（インクルード）文で読み込むようにすることができます。

〈プログラムd〉

ここで宣言する

```
int a, b;
float x, y;
```

ファイル1

```
#include "prg-g.h"
```

ファイル2

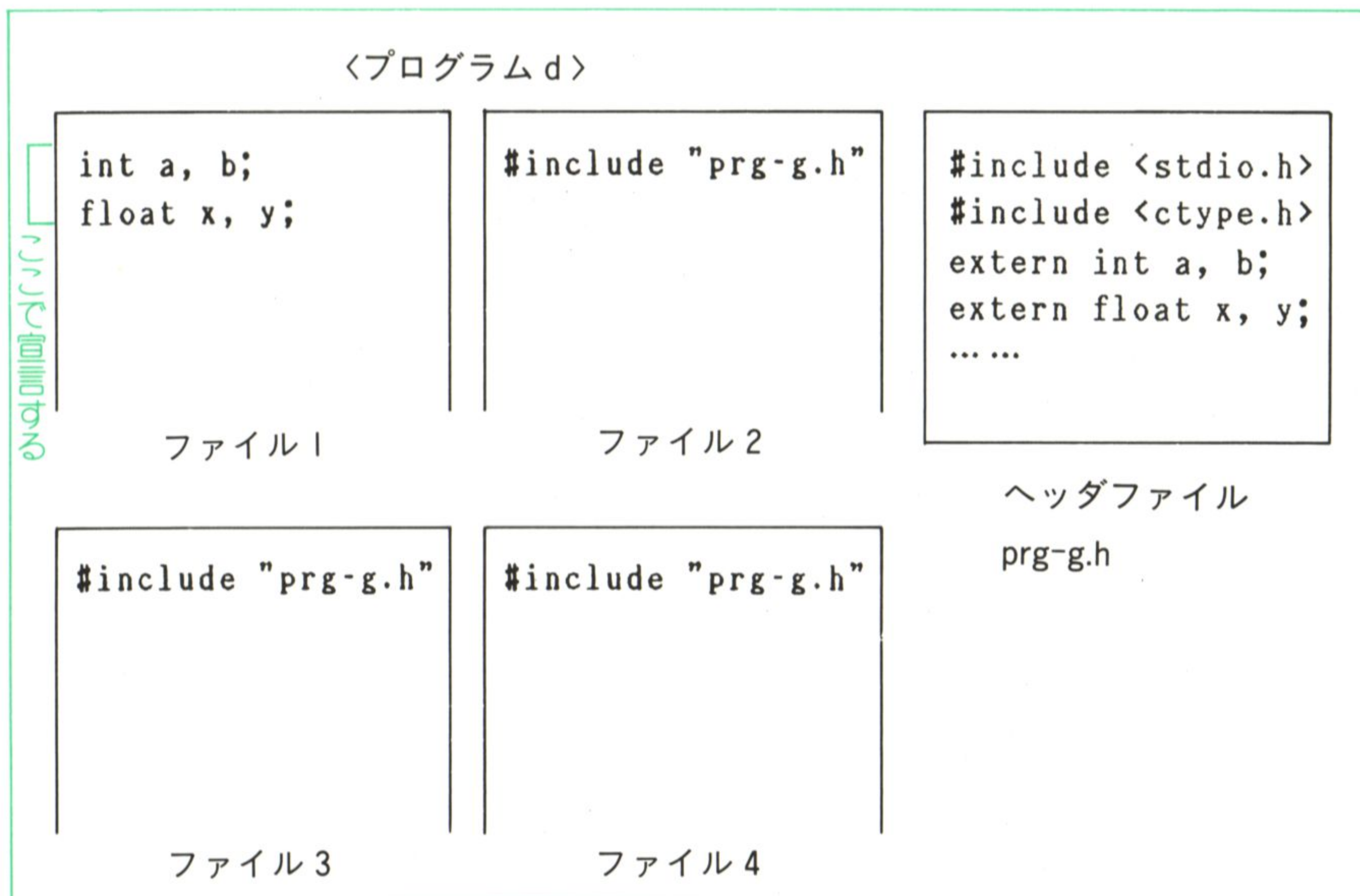
```
#include <stdio.h>
#include <ctype.h>
extern int a, b;
extern float x, y;
……
```

ヘッダファイル
prg-g.h

```
#include "prg-g.h"
```

ファイル3

```
#include "prg-g.h"
```

ファイル4

★注意　ファイル名を囲む記号が「"　　"」の場合はソース・プログラムと同じ場所にヘッダファイルを保存し、「〈　〉」の場合はディレクトリ include の中にヘッダファイルを保存するというのが一般的な約束ごとですが、コンパイラによっては若干異なることがあります。

ふろく2

関数の型宣言の方法

　関数の定義は141ページの例で見たように、

```
int nengou(a, b)
char a;
int b;
{
    ……
}
```

のようにして行われますが、新しく規定されたANSIの規格では、次のように、関数の型と引数(ひきすう)とを1行で宣言することができるようになりました。

```
int nengou(char a, int b)
{
    ……
}
```

　また、ファイルの先頭やヘッダファイルの中でまとめて関数宣言を行うときは、以前は関数の型だけを宣言していましたが、ここでも同じように、

```
int nengou(char a, int b);
void sort(int nmax, int x[]);
```

などと宣言することができます。

■ふろく3

C言語の主な標準関数

C言語で用意されている、主な標準関数をまとめておきます。標準関数はライブラリの中に登録されており、通常「LIB」というディレクトリの中にまとめられています。これらを使用する場合は、関数の型などを宣言している各ヘッダファイルを、プログラムの先頭でインクルードしなければなりません。主な標準関数のヘッダファイルには、次のようなものがあります。

◉主なヘッダファイル

<stdio.h>	標準入出力関数
<ctype.h>	文字クラステスト
<string.h>	文字列関数
<math.h>	数学関数
<stdlib.h>	数値変換と記憶割り当て
<assert.h>	プログラムに診断機能を付加する
<float.h>	浮動小数点演算に関する定数を定義
<stdarg.h>	可変引数リスト
<limits.h>	整数型のサイズを表す定数を定義
<setjmp.h>	非局所的ジャンプ
<signal.h>	外部機器からの割り込みや実行エラーを処理
<time.h>	日付と時刻を扱う
<dos.h>	OSに関連する定義を行う
<errno.h>	エラーコードを定義する

〈stdio.h〉標準入出力関数

種類	書　式	
書式つき入出力関数		
printf	printf（"書式"，変数，……）	
fprintf	fprintf（fp，"書式"，変数，……）	
sprintf	sprintf（配列，"書式"，変数）	
scanf	scanf（"書式"，ポインタ，……）	
fscanf	fscanf（fp，"書式"，ポインタ，……）	
文字入出力関数		
fgetc	変数＝fgetc（ファイルポインタ）	
fgets	fgets（文字型配列，個数，ファイルポインタ）	
fputc	fputc（ファイルポインタ）	
fputs	fputs（文字型配列，ファイルポインタ）	

機　能	使用例
指定された書式で、数字や文字を標準出力装置(画面)へ出力する	printf（"a＝%f, b＝%f", a, b)； ……画面へ変数 a、b の値を表示
指定された書式で、数字や文字をファイルへ出力する	fprintf（fp, "%s", s)； ……fp で指し示すファイルへ配列 s の内容を出力
別の文字列へ、書式を変えてコピーする	sprintf（s, "%d%d", a, b)； ……整数値 a、b を文字型に変更して文字配列 s に入れる
標準入力装置(キーボード)から指定された書式で、文字や数字を入力する	scanf（"%c%c", &a, &b)； ……変数 a、b にキーボードから 1 文字ずつ入力
ファイルから指定された書式で、文字や数字を入力する	fscanf（fp, "%d%d", &a, &b)； ……fp で指し示すファイルから変数 a、b へ整数値を入力
ファイルから 1 文字入力する	a＝fgetc（fp)； ……fp で指し示すファイルから 1 文字入力し、a に代入する
ファイルから最大n−1文字を入力する	fgets（s, n, fp)； ……fp から文字配列 s に n−1文字入力する
ファイルへ 1 文字出力する	fputc（fp)； ……fp で指し示すファイルへ 1 文字出力する
ファイルへ文字列を出力する	fputs（s, fp)； ……fp へ、文字配列 s の内容を出力する

種類	書　式
getc	変数＝getc（ファイルポインタ）
getchar	変数＝getchar　()
gets	gets（文字型配列）
putc	putc（文字型変数，ファイルポインタ）
putchar	putchar（文字型変数）
puts	puts（文字型配列）

〈ctype.h〉文字クラステスト

Cはint型、返値はすべてint型

種類	機　能	使用例
tolower(c)	小文字に変換する	int a, b; a = 'A' b = tolower(a); └bには小文字aが入る
toupper(c)	大文字に変換する	int a, b; a = 'a' ; b = toupper(a); └bには大文字Aが入る

機　能	使用例
指定した入力装置から1文字入力する	a＝getc (fp)； ……ファイル fp から1文字入力して a に代入
標準入力装置(キーボード)から1文字入力する	a＝getchar ()； ……キーボードから入力した文字を変数 a に代入
標準入力装置(キーボード)から文字列を入力する	get (s)； ……キーボードから1行ぶん文字を入力して配列 s に入れる
指定した装置へ1文字出力する	putc (a, fp)； ……fp で指し示すファイルへ変数 a の内容を1文字出力
標準出力装置(画面)へ1文字出力する	putchar (a)； ……画面へ変数 a の内容を1文字出力
標準出力装置(画面)へ文字列を出力する	puts (s)； ……画面へ文字配列 s の内容を出力

種類	機　能	使用例

isdigit(c)	10進数の判定	int a, b, c;
		a = 123;
		if (isdigit(a) = = EOF)
		printf (”a は10進数でない”)；

isprint(c)	印字可能文字(スペースを含む)	int a = ’ ’, b;
		if (isprint(a) ! = NULL) {
		……；
		}
		(NULLはstdio.hの中で0と定義されている)

〈string.h〉文字列関数

sa は文字型変数、sb は文字型変数または定数、n はバイト数

種　類	機　能	使用例

char *strcpy (sa, sb)	文字列をコピー	char *s;
		strcpy(s, "Hello");
		printf("%s", s);
		└→sにはHelloが入る

char *strcat(sa, sb)	文字列をつなぐ	char *a, *b;
		a = "Good";
		b = "Morning";
		strcat(a, b);
		printf("%s", a);
		└→GoodMorning

char *strncat(sa, sb, n)	sb から n 文字を sa の終わりにつなぐ	char *a, *b;
		a = "Good";
		b = "byname";
		strncat(a, b, 2);
		printf("%s", a);
		└→Goodby

int strcmp(ca, cb)	caとcbのアスキーコードを比べる
	cs=cb → >0
	ca=cb → =0
	ca<cb → <0
		char *a, *b, c;
		a = "Go";
		b = "Do";
		if (strcmp(a, b) >0) {
		……
		}

⟨math.h⟩ 数学関数

x、y は double 型、返値はすべて double 型

種　類	機　能
sin(x)	x の正弦関数　$\sin x$
cos(x)	x の余弦関数　$\cos x$
tan(x)	x の正接関数　$\tan x$
asin(x)	x の逆正弦関数　$\sin^{-1} x$　$[-\pi/2, \pi/2]$,$x \in [-1, 1]$
acos(x)	x の逆余弦関数　$\cos^{-1} x$　$[0, \pi]$,$x \in [-1, 1]$
atan(x)	x の逆正接関数　$\tan^{-1} x$　$[-\pi/2, \pi/2]$
sinh(x)	x 双曲線正弦関数　$\sinh x$
cosh(x)	x 双曲線余弦関数　$\cosh x$
tanh(x)	x 双曲線正接関数　$\tanh x$
exp(x)	指数関数　e^x
log(x)	自然対数　$\ln x$
log10(x)	対数関数　$\log_{10} x$
pow(x, y)	べき乗　x^y　$x>0$、または $x<0$ で $y=$整数のとき
sqrt(x)	平方根　$\sqrt{x}$
fabs(x)	絶対値　$\|x\|$
fmod(x)	剰余　mod x

⟨stdlib.h⟩ 数値変換と記憶割り当て

s は文字列変数

種　類	機　能
double atof(s)	文字列 s を double 型に変換
int atoi(s)	文字列 s を int 型に変換
long atol(s)	文字列 s を long 型に変換
int rand( )	乱数を返す
void *calloc(n, m)	m バイト×n 個の領域を割り当てる
void *malloc(n)	n バイトの大きさの領域を割り当てる
void free(void *p)	ポインタ p が指す領域を解放する
void abort( )	プログラムを異常終了する
void exit(const)	プログラムを終了する
int abs(int n)	int 型の変数 n の絶対値
long labs(long n)	long 型の変数 n の絶対値

■ふろく4

エスケープ符号列

本文の中で、「¥0」や「¥n」、「¥t」などのように「¥」記号のつけられた文字が出てきました。この「¥」記号は日本だけのもので、同じ符号がアメリカでは「＼」（バックスラッシュ）に対応しているものですが、「¥」のついた文字は、実はそれぞれが1バイト文字コードを書き表すC言語特有のものです。「¥」のついた文字コードはエスケープ符号列（エスケープシーケンス）と総称されるもので、次の表にあるのがそのすべてです。

種類	意　味	機　能
¥a	警告（ベル）	ベルを鳴らす
¥b	バックスペース	1文字ぶん左へ
¥f	改ページ	ページを変える
¥n	改行	次の行の先頭へ
¥r	復帰	同じ行の先頭へ
¥t	水平タブ	水平方向のタブ
¥v	垂直タブ	垂直方向のタブ
¥¥	「¥」を示す	——
¥?	「?」を示す	——
¥'	「'」を示す	——
¥"	「"」を示す	——
¥ooo	8進数	¥123＝8進数123
¥xhh	16進数	¥xff＝16進数 ff

「¥ooo」は任意の1バイトのビットパターンを表し、「¥」に続いて「ooo」の部分に3桁以内の0～7が書かれ、「¥0」はこの内の1例です。「¥xhh」は同じく任意の1バイトのビットパターンを16進数で表す場合の書きかたで、「¥x」に続いて「hh」の部分に2桁の0～9、a～f（またはA～F）が入ります。

用語さくいん

太数字は、その用語が見出しになっていることを表す。

●記号

●アルファベット

●ア行

●カ行

●サ行

●タ行

●ナ行

●ハ行

●マ行

●ラ行

紹介市販ソフト

※以下は、当該各社の商標または商品名です。

●著者略歴

JTM企画株式会社／Writer荒瀬　遙（あらせ　はるか）
1986年3月、桃の節句に会社設立。
『新・一太郎』（西東社刊）を皮切りに、パソコン関係の書籍を多数世に出している。
主な業務は、コンピュータのマニュアル制作、翻訳およびパソコンソフトの開発。主な著書は次のとおり。
『一太郎Ver.3／Ver.4』、『初めての人のC言語入門』『花子Ver.2』『よくわかる一太郎dash』（以上西東社）、『TURBO C Ver.2.0独学のすすめ』（翻訳：マグロウヒル出版）、『一夜漬MIFES-98 Ver.4』『一夜漬Quick BASIC』『一夜漬Lotus 1-2-3 Plus』（以上、ラジオ技術社）『アシストワードがよくわかる本』『アシストカルク　関数・マクロ徹底活用マニュアル』（以上、HBJ出版局）

初めての人のC言語入門

著　者　JTM企画株式会社／荒瀬　遙　　1993年11月30日　発行

発行者　若　松　親　光

発行所　株式会社　西東社（せいとうしゃ）
〒101　東京都千代田区神田錦町3-15
☎　東京（03）3291－5815
FAX　東京（03）3233－2466
振替口座番号／東京　8－6499

ISBN4-7916-0877-1

読者の皆さまへ

弊社の出版物は、全国有名小売書店の店頭に備えてあります。実物をごらんになって、お買い求めください。

書店に品切れの節は、店員にご注文いただければ、弊社から取り寄せてお渡しいたします。

ぜひ、お近くの書店をご利用ください。

株式会社 西東社（せいとうしゃ）

東京都千代田区神田錦町３丁目15番地
電話　東京（03）３２９１－５８１５番

❖西東社は、多くの人々に親しまれ、日々の生活に潤いと指標をあたえる本づくりをめざしています。

受験用

情報処理技術者試験用語辞典

●日本ナレッジインダストリ株式会社・編著

本書は、「第２種情報処理技術者試験」の受験者を主な対象に、特に受験直前の知識の整理、学習の総しあげに利用できるようにまとめた用語辞典です。

〈本書の特長〉

❶重要な用語だけに絞っているので、知識の整理がしやすくなっています。

❷不得意な分野、確認しておきたい分野を重点的に学習できるよう、内容によって全用語を６分野に分け、それぞれをさらに項目分けしました。

❸用語の意味、英文、同義語、略語のフルスペルなど、各用語の必須事項を一か所にまとめたので、ひと目で確認でき、頭に入れやすいようになっています。

❹各用語のくわしい解説は、簡潔を第一としてまとめています。

❺各分野ごとに練習問題をつけ、学習の効果をチェックできるようになっています。

❻簡便な用語集としても利用できるように、巻末に詳細なさくいんをつけました。

以上のように本書は、知識の整理が効率よく行えるようにさまざまなくふうをこらした「第２種情報処理技術者試験」参考書の決定版です。

最新版

第2種情報処理技術者試験問題集

●東京都立科学技術大学教授　**大槻繁雄・監修**

本書は、「第２種情報処理技術者試験」受験者を対象に、実際出題された問題、最低限必要な知識を満載した書です。

〈本書の特長〉

❶過去に出題された問題から、各科目の代表的な70問を厳選し、解答方法を示すと同時に、各設問に関する要点を整理しています。

❷出題範囲の大部分をカバーするハードウェア、ソフトウェア、関連知識、およびプログラム作成の流れ図についてとりあげました。

❸毎回似たような形で必ず出題されているものについては、「基礎知識」としてまとめ、また、最近出題されることの多い「計算機システムと通信」についても別項にまとめています。

❹巻末に、実力を試す目的で、各科目の出題割合に合わせた模擬試験形式の問題集をつけました。

❺とりあげた70問については、基本的に「解答のヒントとポイント」「解きかた」「要点の整理」「関連事項」「解答」の順に解説しました。

最短にして、最高の合格をものにしたいという読者に、是非おすすめしたい問題集です。